OpenInsight Git User Guide

Version 1.0

REVELATION
SOFTWARE
A Division of Revelation Technologies, Inc.

Part No. 214-968

Printed in the United States of America.

Table of Contents

Introduction

OpenInsight Git is an interface from OpenInsight to the Git source code management system. It is included in OpenInsight 10.0 and above.

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

You can read more about Git on its web site at http://www.git-scm.com

For many, source code management is a black art reserved for large corporations who have hundreds of developers. You may think that source code management systems are complicated, involve many steps that have nothing to do with writing code and generally get in the way of being productive. And, for the most part, you would be right!! OpenInsight Git has been designed to be as painless and transparent as possible. There are rules that you will need to follow, but, for the most part, there are many benefits to using source code management software. For example:

1. Backup of your source code
2. Version control
3. Change control
4. Collaboration with developers remote to your location

OpenInsight Git was originally designed to integrate OpenInsight's development tools with the Git source code management software (SCM). However, there is no reason why OpenInsight Git can't work with other SCM's like Subversion or Mercurial.

OpenInsight Git's philosophy is simple. It mirrors your OpenInsight source code, stored procedures, windows, messages, help, etc. as text files in Windows directories. From there, your chosen source code management system does the rest.

This manual consists of a quick start guide and a more detailed user manual. The quick start guide is designed to get you up and running in the shortest amount of time using the EXAMPLES application. The chapter "The Seven Steps to Using OpenInsight Git" goes into more details about each step.

It is not this guide's intention to teach you all the ins and outs of Git or of any other SCM. Git comes with many levels of documentation. We highly recommend a book called Pro Git which is the de facto bible on the subject.

Most of you will only use a fraction of the features that Git or any SCM offers. And that is ok. The important thing is to start using an SCM as soon as possible. From a one man developer to a team of tens or hundreds of developers, using an SCM should become part of your daily work routine.

What you will need to use OpenInsight Git

The following software is required to use OpenInsight Git

- OpenInsight v10.0 or above
- Git Client Software – See below for details

OpenInsight v10.0 introduces a feature that enables OpenInsight Git to fully integrate into the OpenInsight repository. Any change saved in an OpenInsight development tool such as the Editor ++, the Form Designer, BRW or O4W will automatically be pushed to the local Git repository, making the source management process much easier.

Installing Git

Git comes with built-in GUI tools (git-gui, gitk), but there are several third-party tools for users looking for a specific experience.

You can download Git from the following website: http://git-scm.com/downloads

A list of third party Git clients is available here: http://git-scm.com/downloads/guis

A popular Git client is Git Extensions. We recommend using Git Extensions with OpenInsight Git. You can download Git Extensions from:

http://sourceforge.net/projects/gitextensions/

Whether you install the standard Git client or choose a third party Git client, you still need to install the basic Git software. Most third party Git clients bundle in basic Git as part of their installation process.

Below are a few tips to follow during the basic Git installation. Please note that your install process may look slightly different to the screen shots below, but the basic settings are the same.

Tip! *Git Extensions starts a main installation program, which then starts individual installation programs for Git and its default difference software, KDiff. Sometimes, these individual installation program windows appear behind the main Git Extensions installation window. You many need to Alt-Tab to see the individual installation window and continue the installation process.*

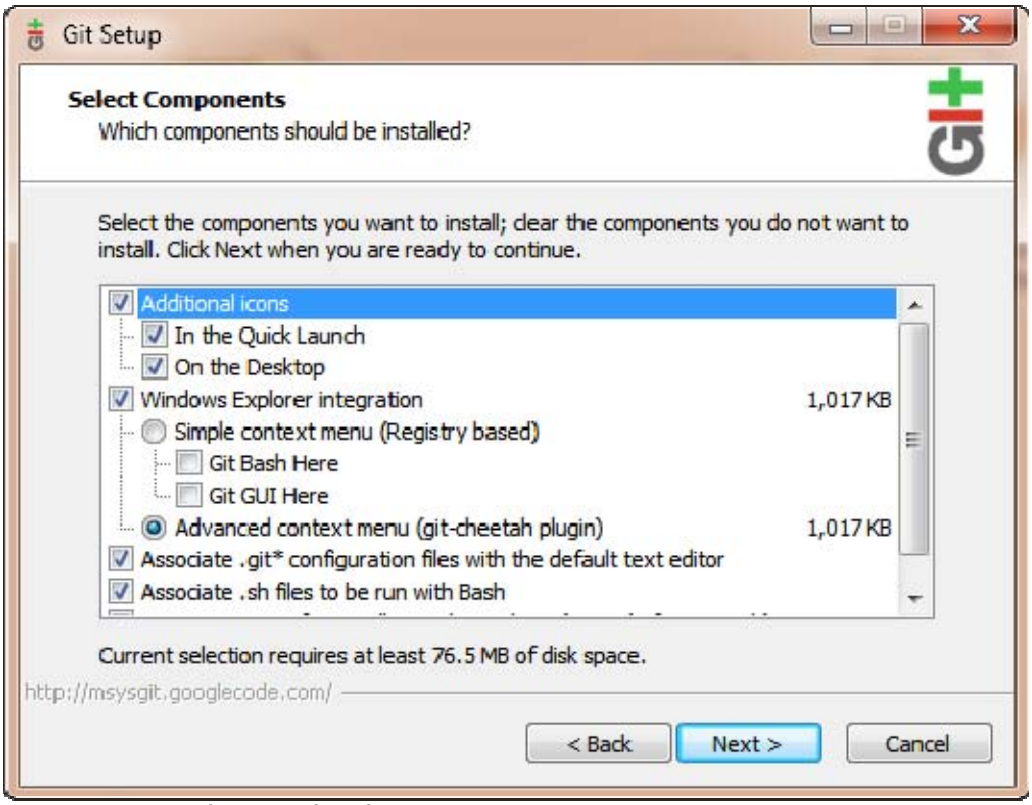

Figure 1 - Git options Screen 1

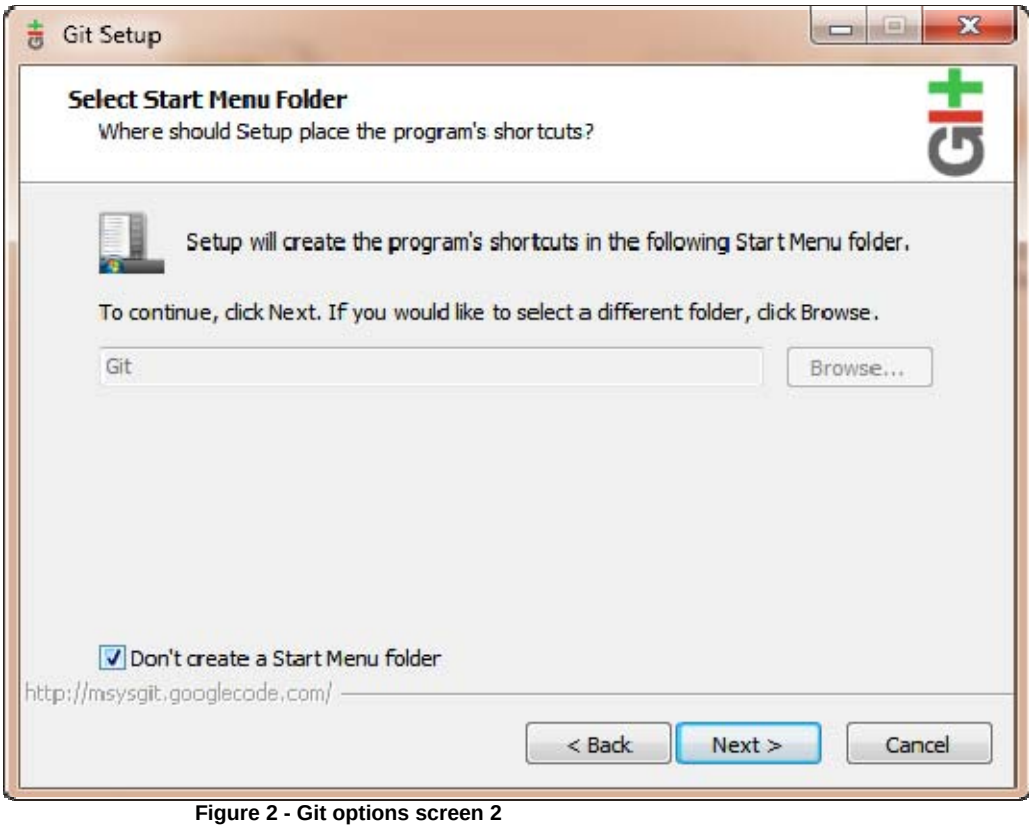

Figure 2 - Git options screen 2

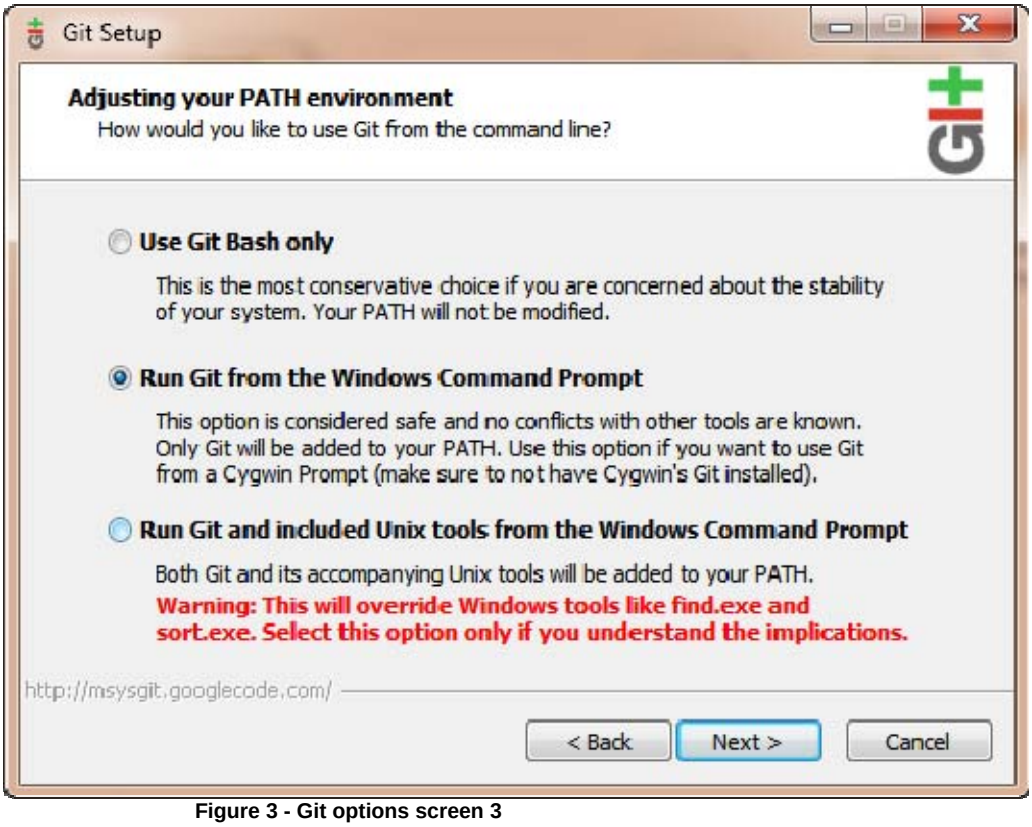

Figure 3 - Git options screen 3

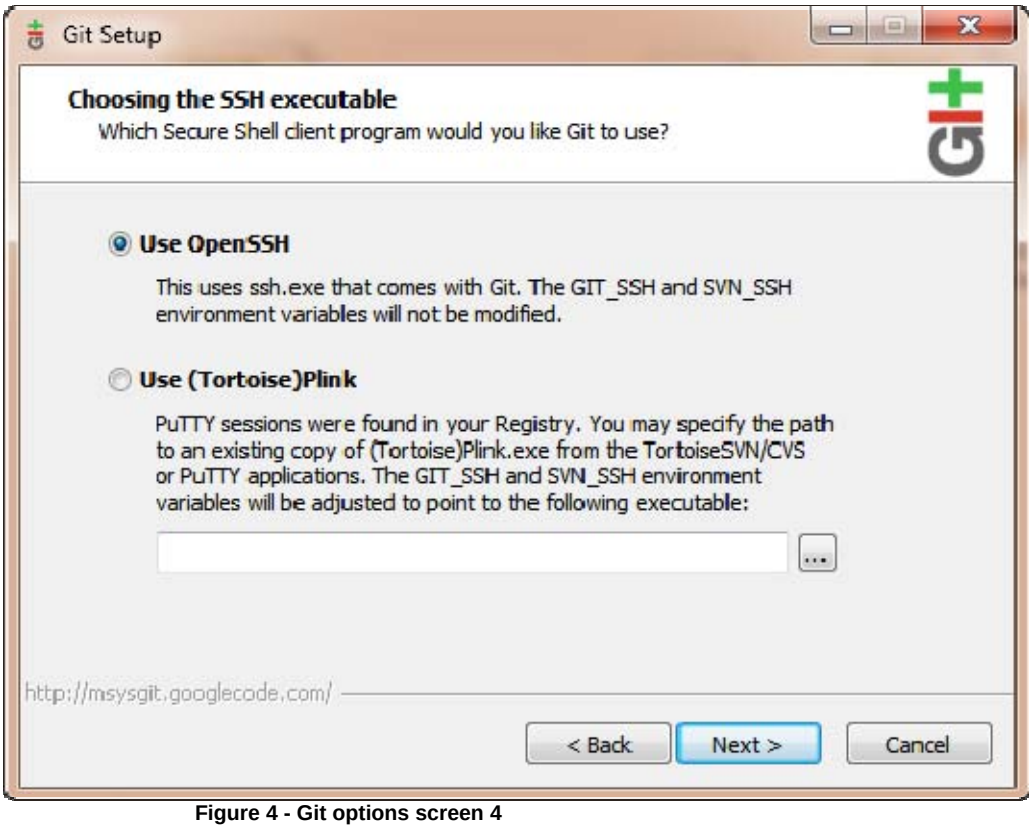

Figure 4 - Git options screen 4

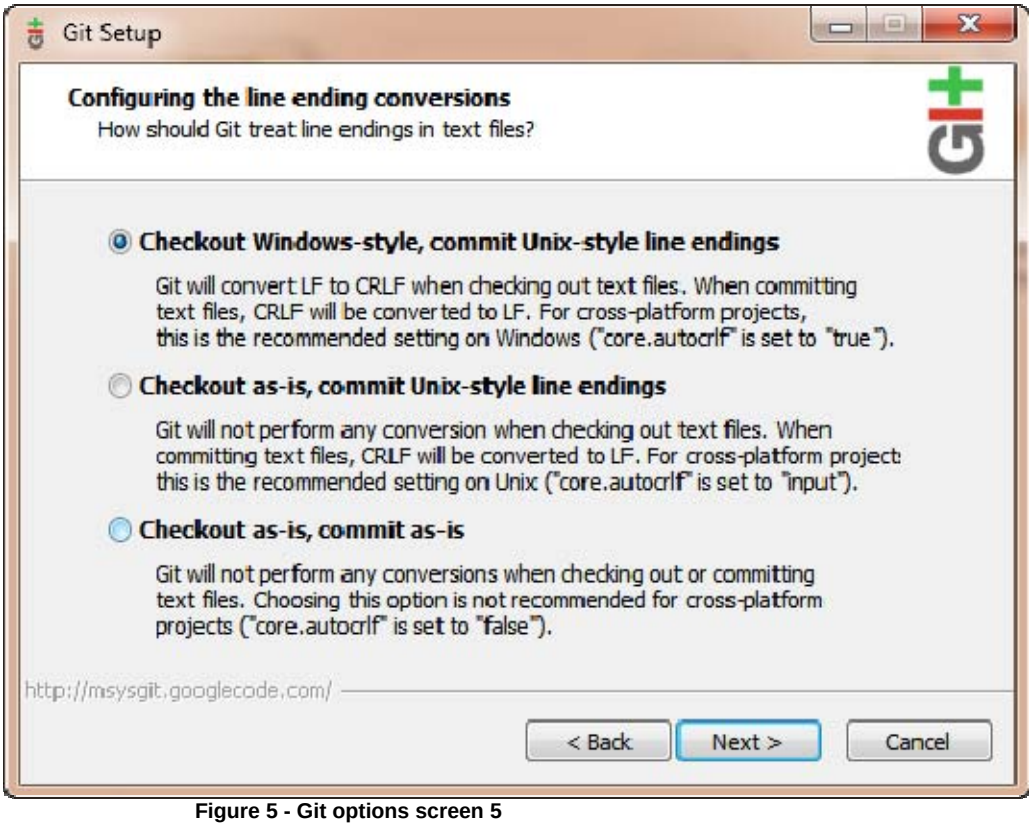

Figure 5 - Git options screen 5

Initializing a Git Repository

The next step is to initialize a Git repository. To do this, we use Git Extensions which has been added to Windows Explorer context menu.

Open Windows Explorer and navigate to the directory where you wish to create your Git repository. The directory can be anywhere on your local hard drive. In the illustration below, we will create a new Git repository in the C:\Revsoft \Git directory.

These screen shots are for the Git Extensions client. Basic Git and other third party clients offer similar Windows Explorer options.

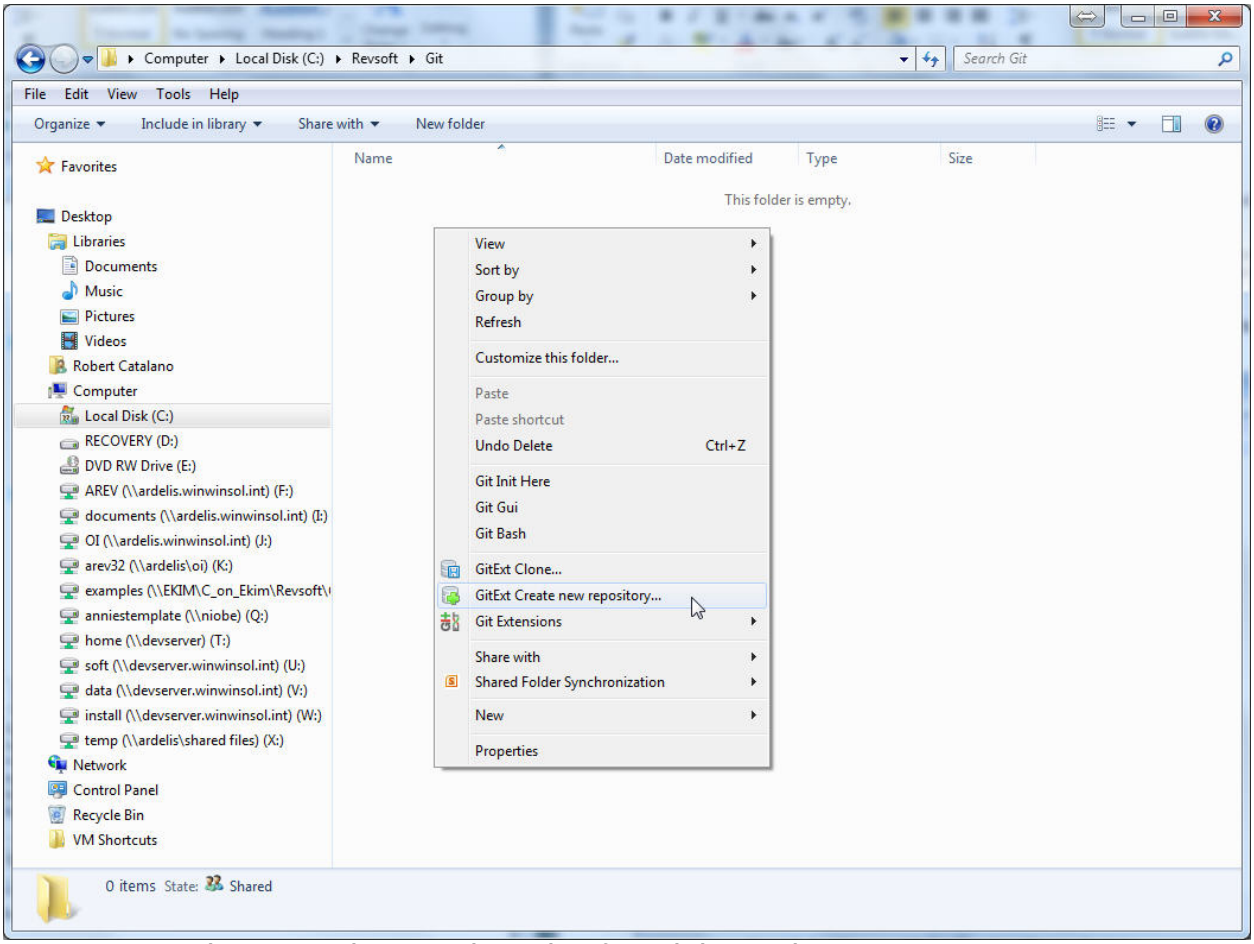

Figure 6 - Creating a new Git repository from Windows Explorer

Right click on the folder which will contain the Git repository and select GitExt Create new repository. The following screen will appear.

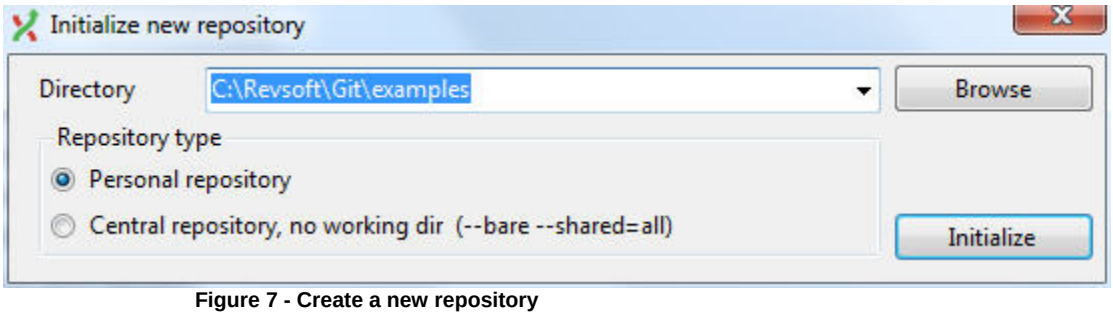

Figure 7 - Create a new repository

In the above example, Git will create a new directory "examples" in the C:\Revsoft\Git directory and will initialize an empty Git repository. *Please ensure that you select "Personal repository" as the repository type.* After you click the Initialize button your C:\Revsoft\Git\examples directory will display the following message:

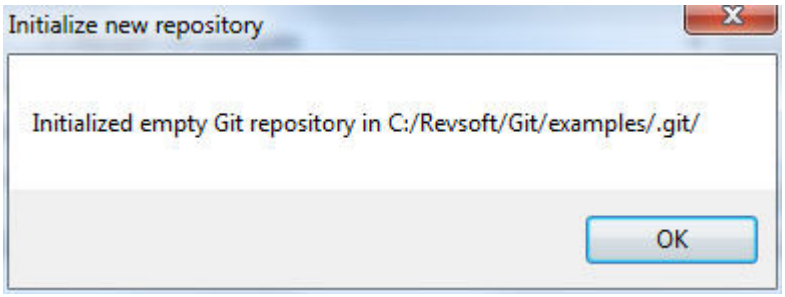

Figure 8 - Git repository initialization message

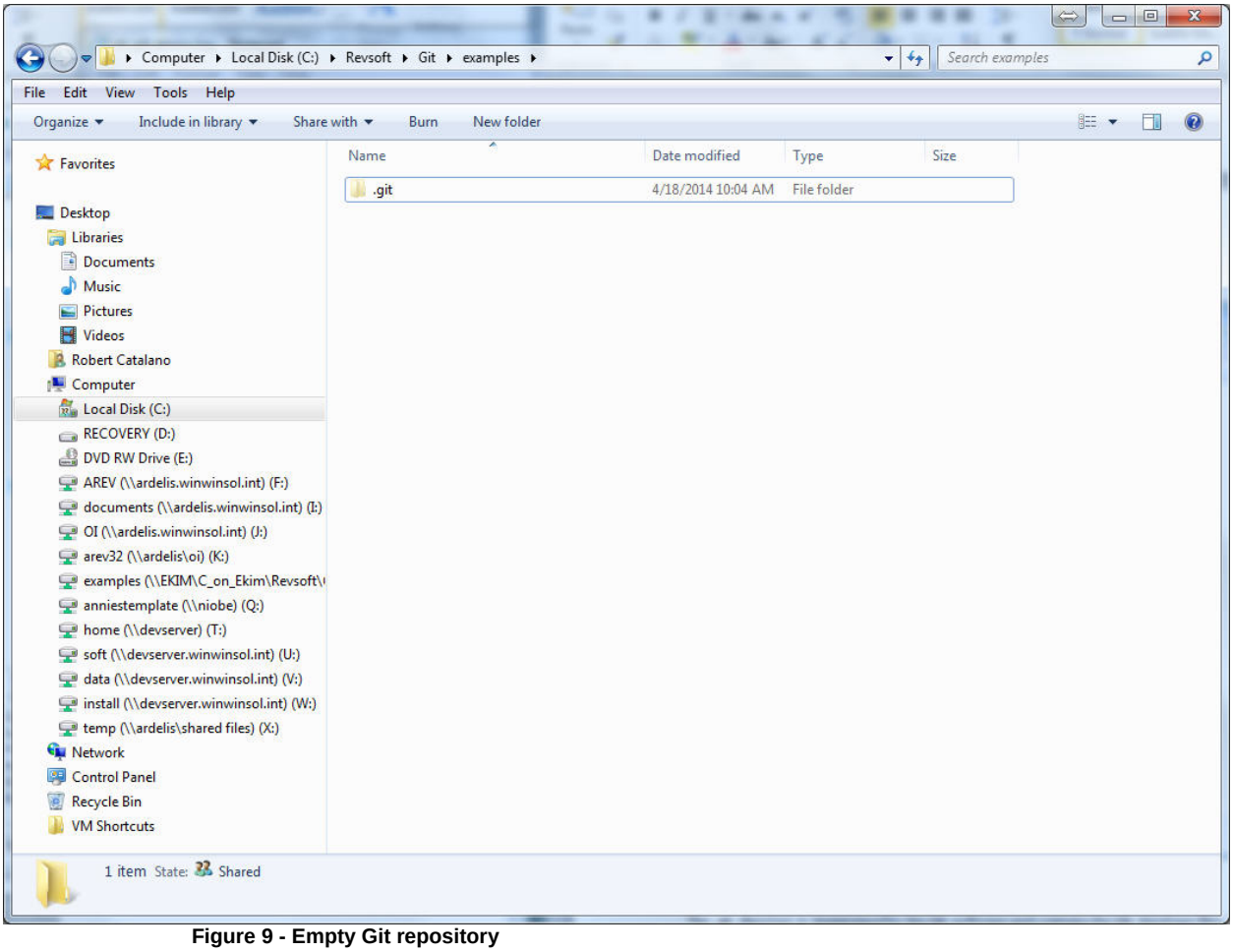

Figure 9 - Empty Git repository

The .git directory is maintained by the Git software and contains the Git database files. You do not need to edit or change any files in this directory. All configuration settings are handled by your chosen Git client software.
You now have an empty Git repository ready for use.

Git Terms

Below is a list of the most common Git terms you need to know.

Term	Description
Working Directory	A view of your local Git repository which contains a list of uncommitted changes to files.
Master Branch	A view of your local Git repository which contains a list of files that have been committed locally or pulled from a remote Git repository
Origin	The name for the repository that the current local Git repository was cloned from.
Pull	Downloads changes from a remote Git repository to your local Git repository.
Merge	Merges changes from a pull into your local Master branch. Some Git UI tools perform a Pull and Merge in one operation.
Push	Uploads committed changes from your local master branch to a remote repository
Diff	Displays the difference between versions of a file, or lists all the changes for a particular commit.
Commit	Moves changes from your local working directory to your local master branch. Each commit should contain a short description and signed off by the local developer. Each commit is assigned a unique id so that it can be identified later.
Blame	A Git tool that identifies who changed a particular line in a file and when it was changed.
+	A new file to be added to your local Git repository – as seen in your "Working Directory"
-	A file that is to be deleted from your local Git repository – as seen in your 'Working Directory"
+++	A new line added to a file – as seen in Git Diff
---	A line removed from a file – as seen in Git Diff
	A new file
	The file has been deleted
	The file has been updated
	The file in Git is identical to the source in OpenInsight

Getting Started

OpenInsight Git is fully integrated with the OpenInsight repository. This means that every time you save changes to an entity in OpenInsight, the changes will automatically be updated to the "working directory" in your local Git repository (OpenInsight v10.0 and above).

The instructions below will guide you through setting up OpenInsight Git for the EXAMPLES application. Step one is to login to the EXAMPLES application in your local copy of OpenInsight.

For a full description of the features of OpenInsight Git please refer to the **Seven Steps to Using OpenInsight Git** chapter.

Setting up OpenInsight Git

The first time you run OpenInsight Git, you will be asked to enter some configuration settings. To start using OpenInsight Git from the Application Manager, select Application Tools, Git, Settings.

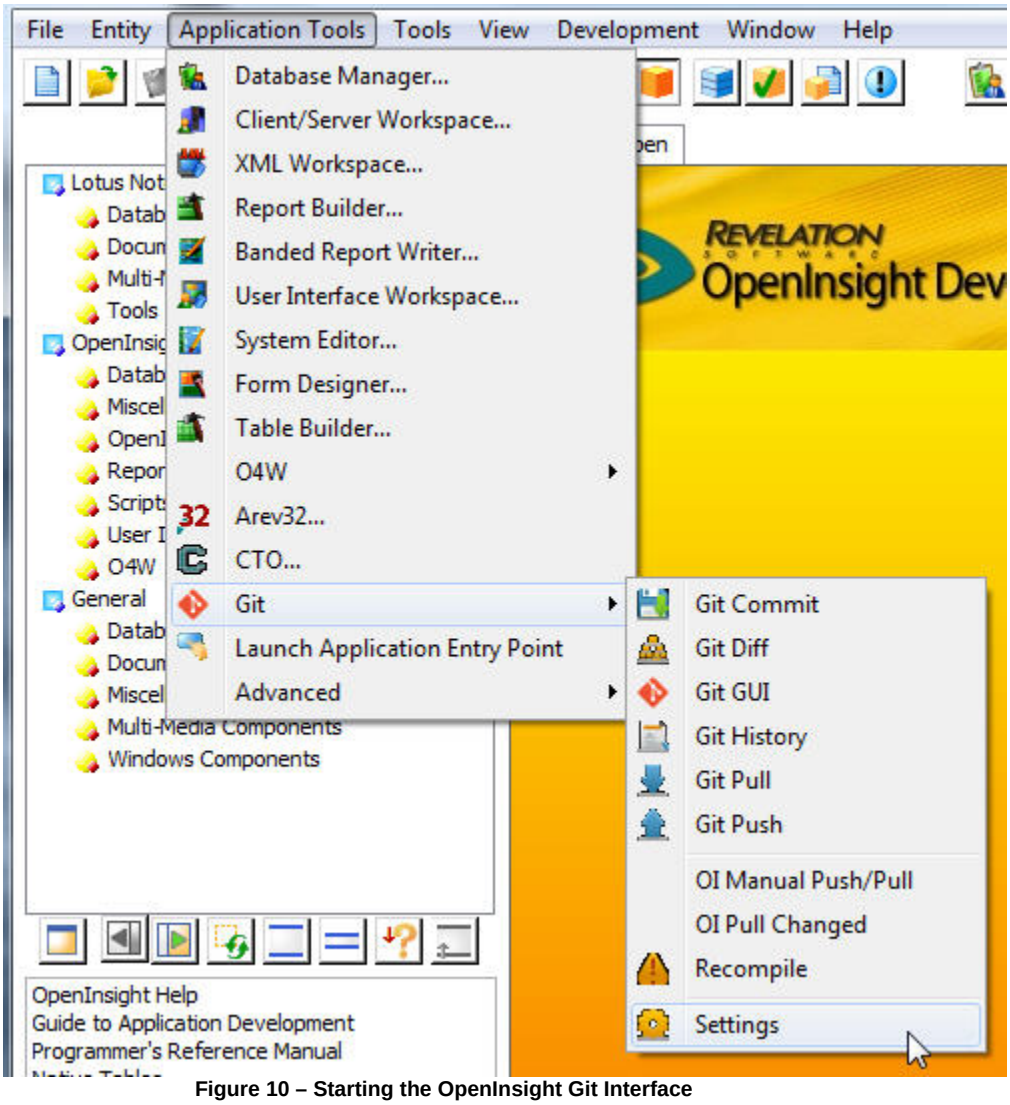

Figure 10 – Starting the OpenInsight Git Interface

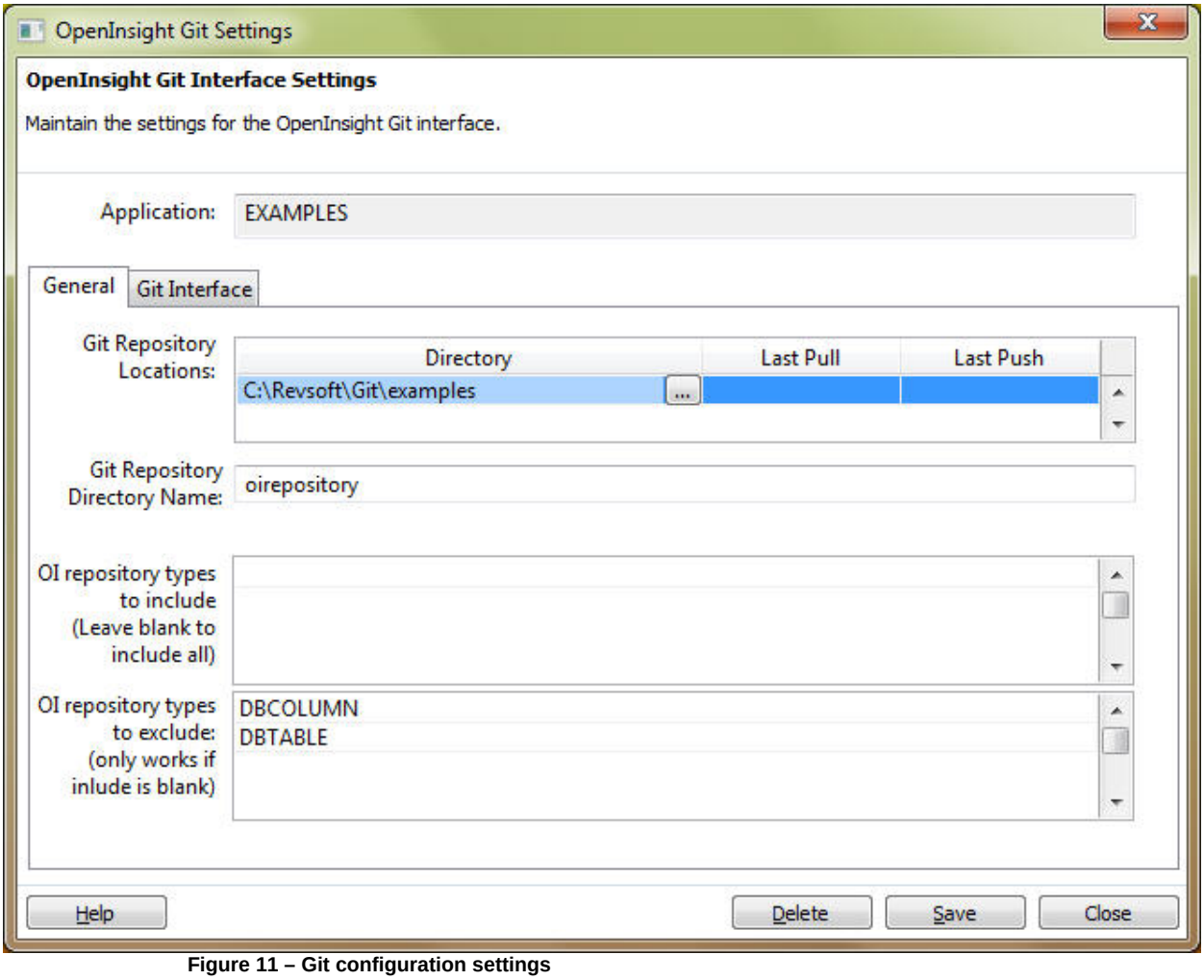

Figure 11 – Git configuration settings

Git Interface Settings Window

The Git Interface Settings window is used to control how OpenInsight and Git interact. The settings for the current OpenInsight application are automatically loaded.

These settings are stored in the SYSENV table with the key of CFG_GIT*appId

Note: OpenInsight Git will only push entities that are flagged as publishable in the OpenInsight repository. If you want to keep some source in your OpenInsight system but not push it out to Git then untick the publishable flag.

General Tab

Git Repository Locations:

Select the directory where your Git repository is located. You can enter multiple locations if you wish. When you push or pull your changes, you can select the location of the Git Repository you want to utilize.

Enter in the path to the directory you created earlier that holds your empty Git repository, C:\Revsoft\Git\examples

NOTE: The first location is used to automatically push changes from OpenInsight to the Git repository via the OpenInsight repository hooks.

Git Repository Directory Name:

Enter in the name you want to use for the directory in Git that stores your OpenInsight source. If this is blank then the default name of "oirepository" is used.

The following replaceable keywords are available here:

%APPID% - replaced with the current application Id
%USERNAME% - replaced with the current OpenInsight username

Use this setting to customize how your OpenInsight source is stored within your Git repository. For example, if you wanted to have one Git repository and store source code for multiple OpenInsight applications in it, then you could change this value to be something like %APPID%\oirepository.

Then your Git repository would have a sub directory for each of your applications like this:

c:\Revsoft\Git\abcco\myapp1\oirepository
c:\Revsoft\Git\abcco\myapp2\oirepository

OI repository types to include:

Select which OI repository types you wish to push out to Git. Enter in the repository types here and only these types will be pushed to Git. If you leave this blank then all repository types will be pushed, with the exception of EXE and DBG types like STPROCEXE and STPROCDBG.

****** Note ******
The following OpenInsight repository types are automatically excluded:

STPROCEXE – stored procedure executable
STPROCDBG – stored procedure debug tables
OIWINEXE – OpenInsight window executable
DBCOMPONENTEXE – database component executable
OIEVENETEXE – OpenInsight event executable
OIEVENTDBG – OpenInsight event debug table

OI repository types to exclude:

Enter in the OI repository types you wish to exclude from being pushed to Git. This value only applies if the "Include" list above is left blank. By default the DBTABLE and DBCOLUMN types are excluded. Revelation Software recommends that the following OI repository types be excluded:

DBCOLUMN
DBTABLE

Git Interface Tab

Contains the default settings are for the Git Extensions client installed in the default location. You will need to change these settings if you are using some other Git client or some other source management software.

Gui:

Enter the path and filename to start your installed Git client UI.

Commit:

Enter the path and filename plus the command to initiate a commit.

Pull:

Enter the path and filename plus the command to initiate a pull from a remote Git repository.

Push:

Enter the path and filename plus the command to initiate a Push to a remote Git Repository.

File History:

Enter the path and filename plus command to open the File History tool for your chosen SCM tool.

Diff Tool:

Enter the path and filename plus the command to open the Diff tool for your chosen SCM tool.

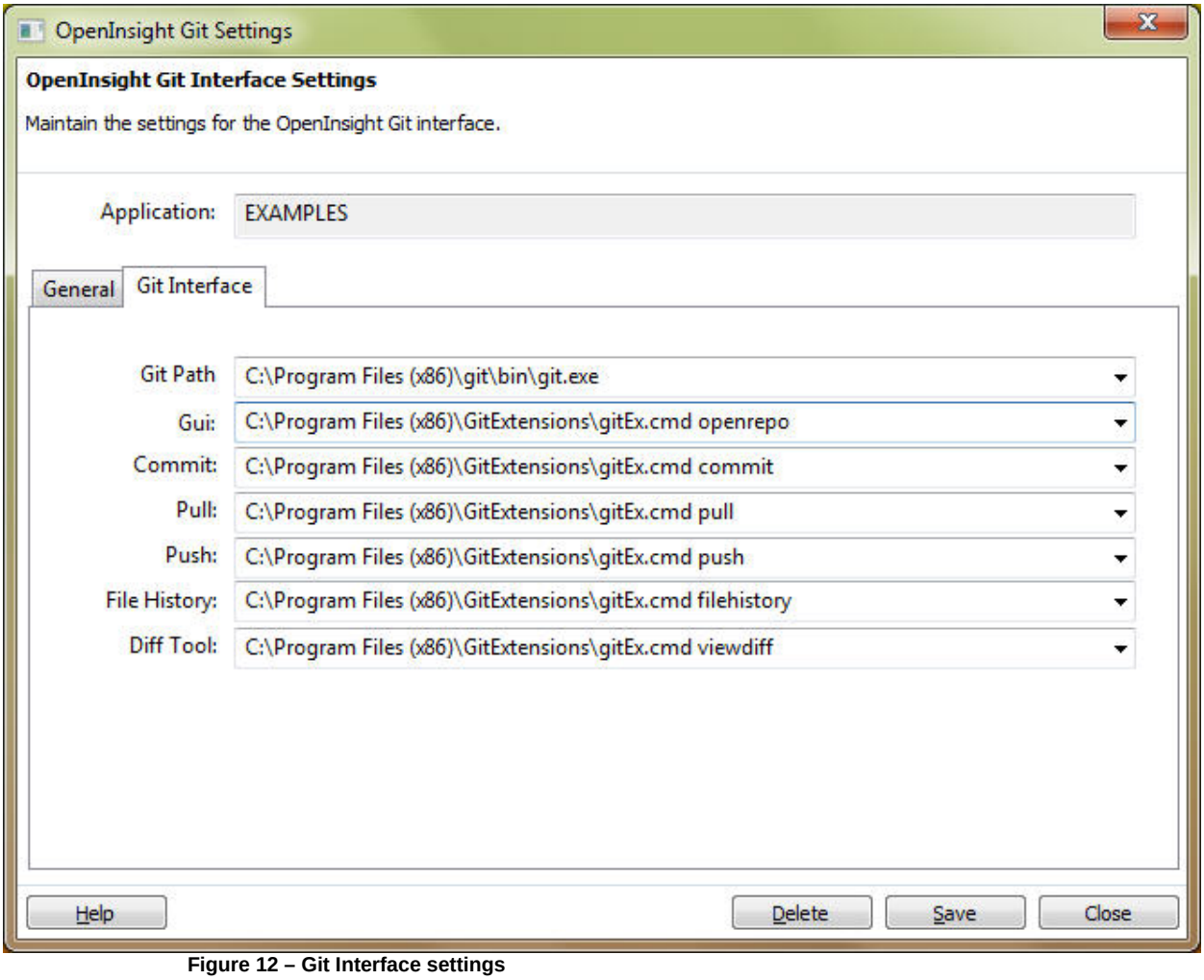

Figure 12 – Git Interface settings

Click the Save button to save your changes and then the Close button to close the settings window. You will be returned to the OpenInsight Application Manager.

Push your source code to your Git Repository

Now you are ready to push your OpenInsight source out to your Git Repository. Your OpenInsight Git window should look something like the following:

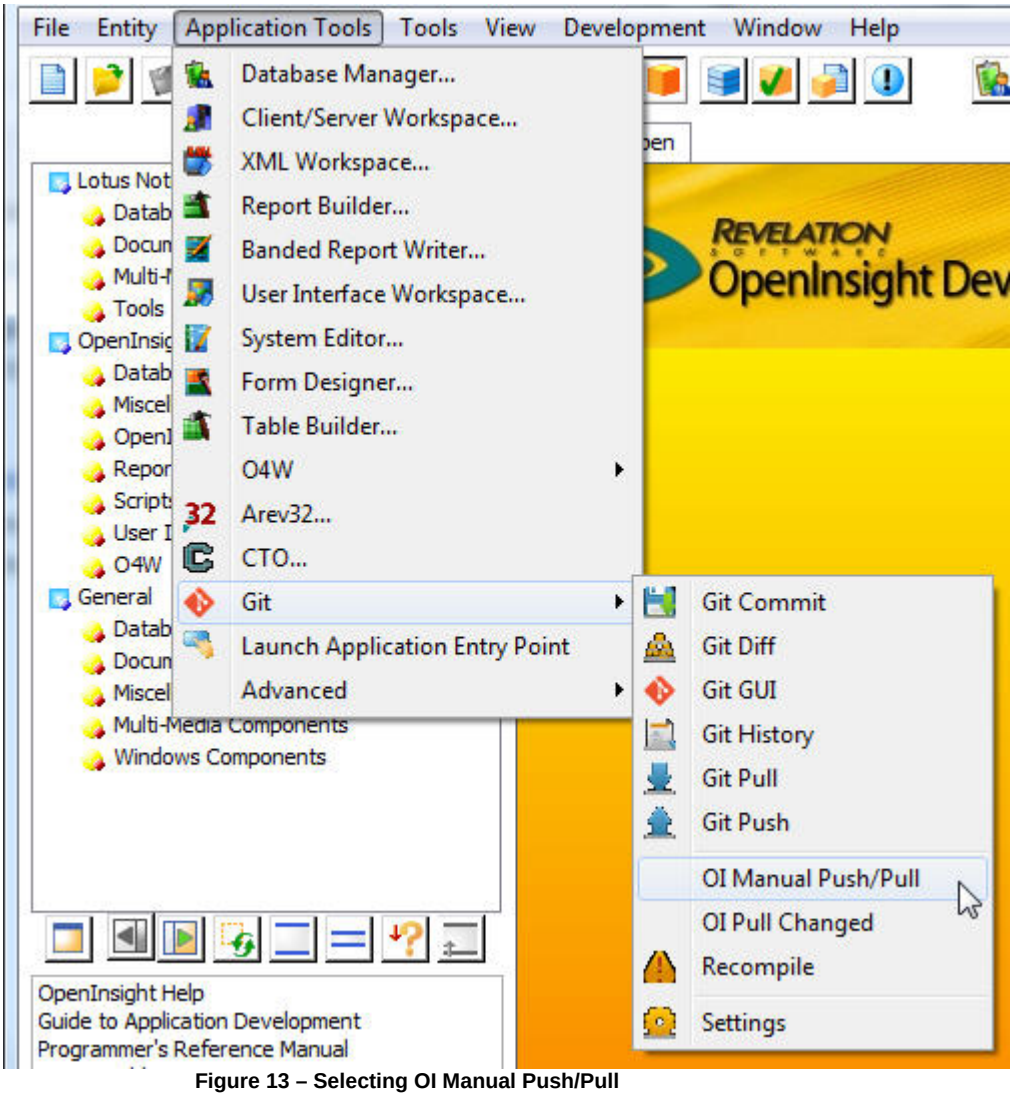

Figure 13 – Selecting OI Manual Push/Pull

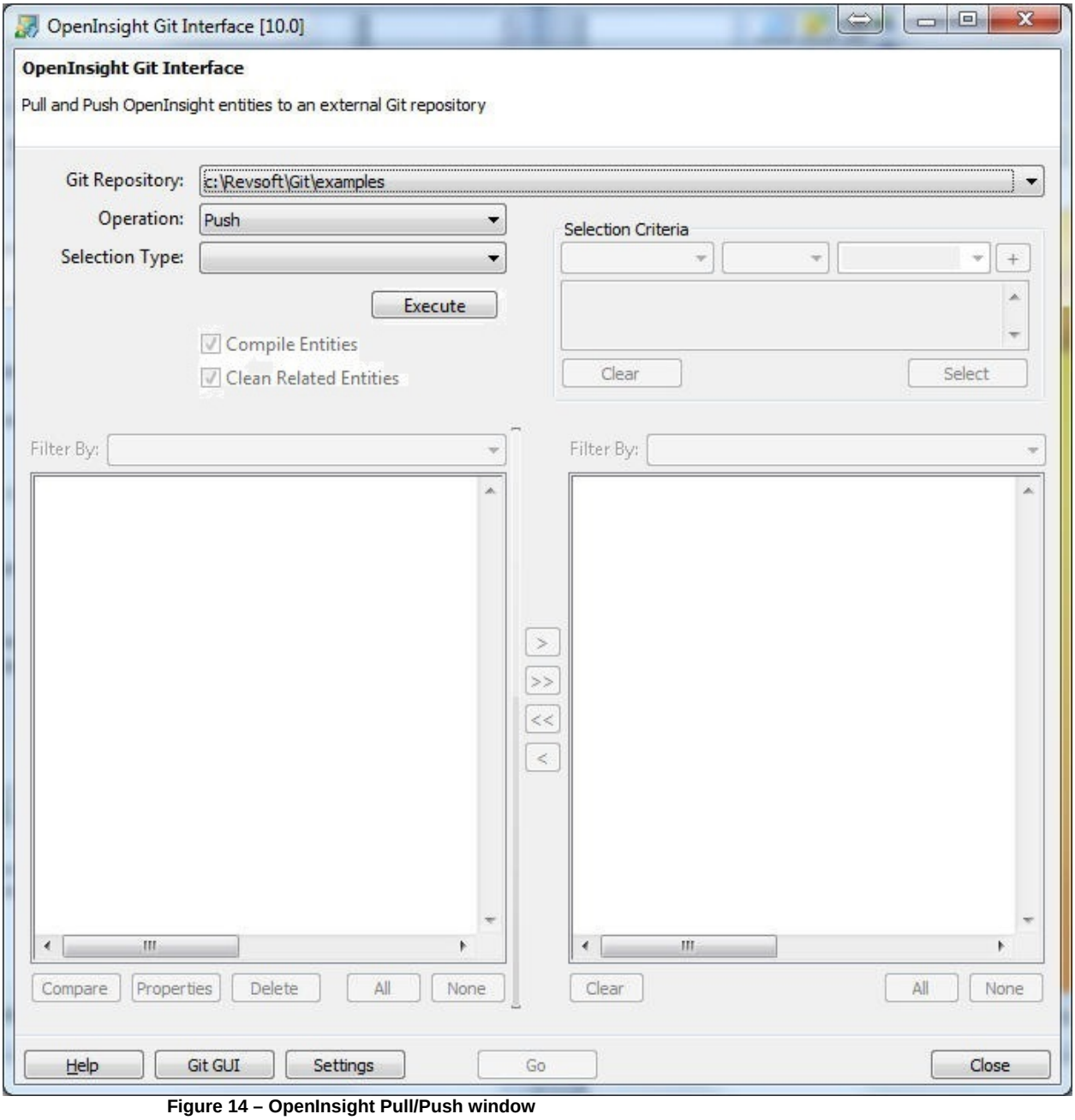

Figure 14 – OpenInsight Pull/Push window

You are about to perform a manual *Push* from OpenInsight to Git. When you do a manual push you can choose which items to push much the same way you choose which items to include in an RDK.

To push your source to the Git Repository:

1. At Operation select “Push”
2. At Selection Type select “All”

OpenInsight Git will build a list of all “pushable” items from your OpenInsight application and display the list in the box on the left**.** You can then choose which items to push by moving the items from the left box to the right box. Items in the right box will be *pushed* to the Git repository thereby overwriting the version in Git.

Select all the items by clicking the >> button. This will move all the items from the left box to the right box. Your window should then look something like this:

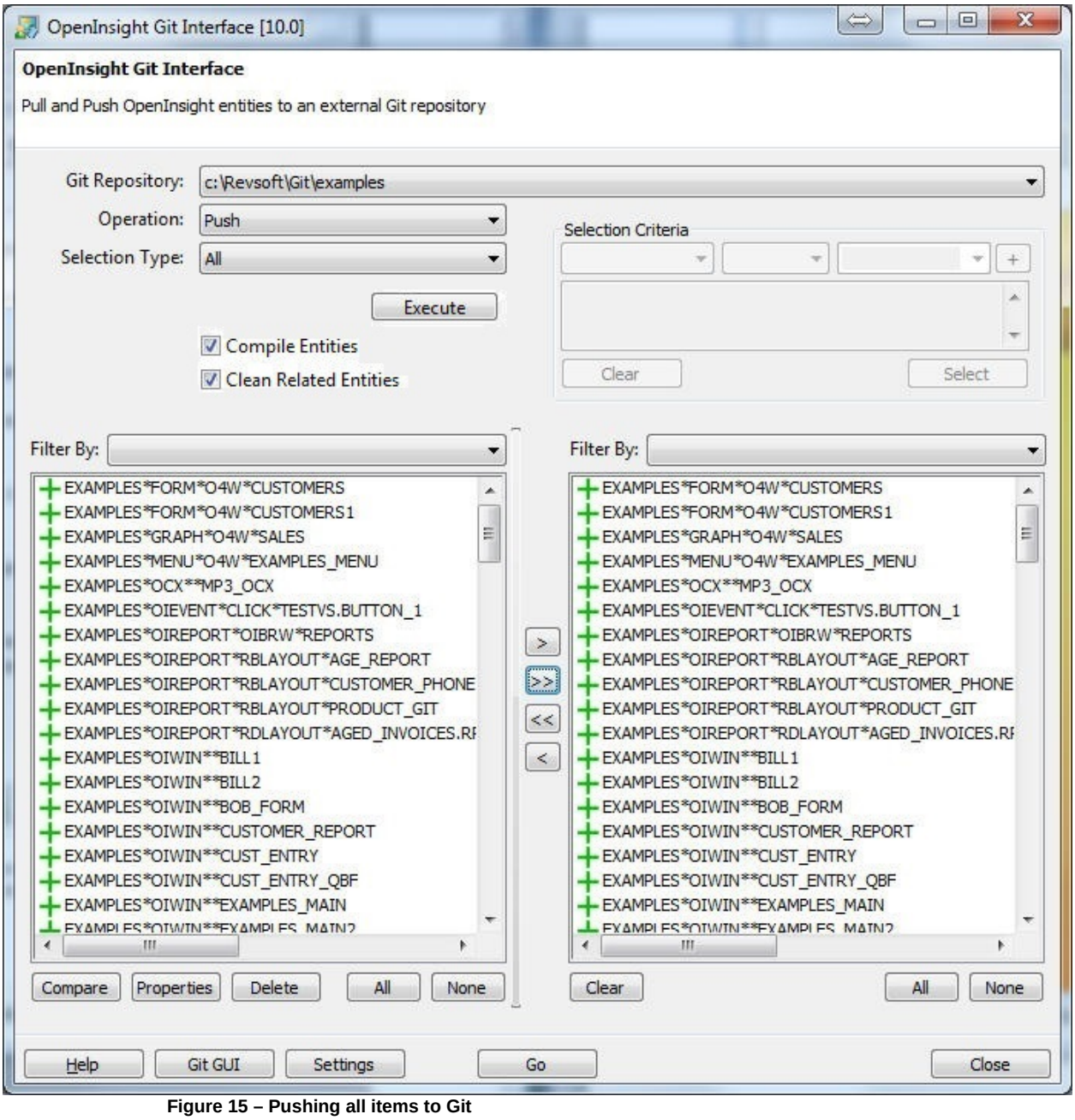

Figure 15 – Pushing all items to Git

When pushing to Git, the green plus icon + means that the item does not currently exist in the Git repository. Since this is your first push from OpenInsight to Git, all the items have a green plus icon.

Now, click the ***Go*** button. OpenInsight Git will push the selected items to your local Git repository.
Note: You will probably see the following message at the end of the push process.

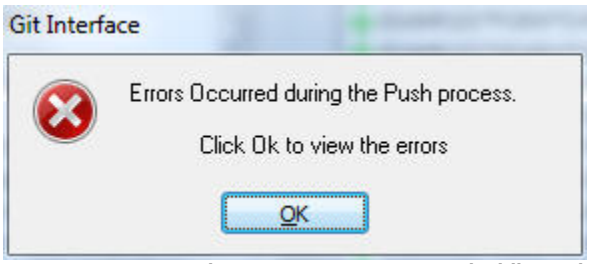

Figure 16 – Errors occurred while pushing to Git

To view the errors, click the OK button. This will open Notepad and display the errors log.

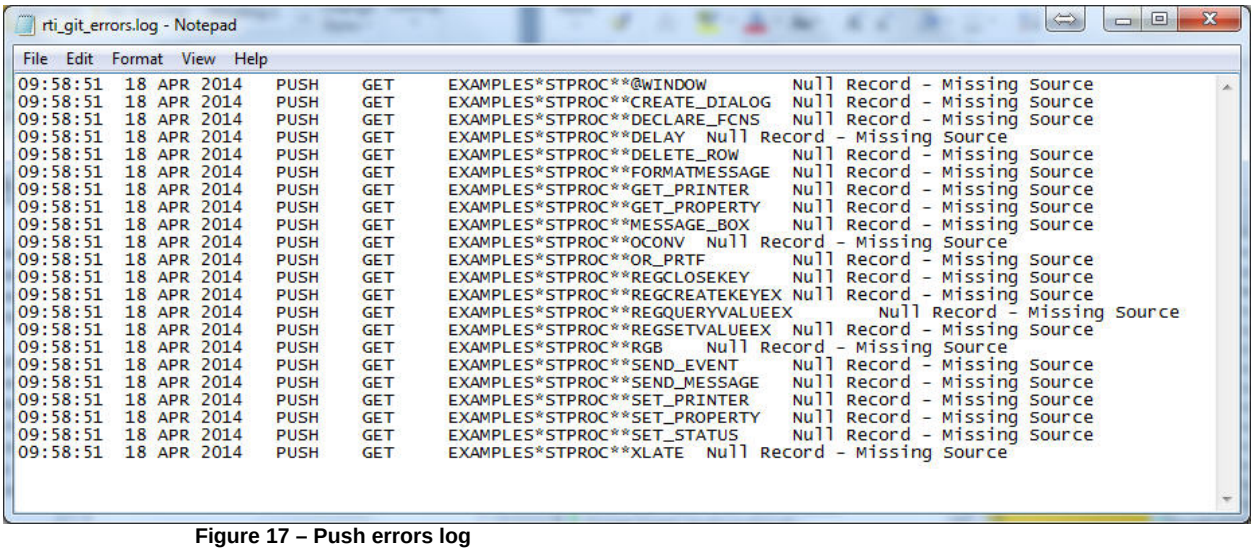

Figure 17 – Push errors log

OpenInsight Git maintains an errors log and a process log in the \logs directory beneath OpenInsight. The error log displays any errors that occur during the last push or pull process. The process log maintains a list of the last 1000 successful push or pull processes.

From the error log above you can see that all these errors occurred because OpenInsight Git found a repository record for some source, but failed to read the source code. This usually means that you have invalid repository records and they need to be cleaned up.

Now if you look at your C:\Revsoft\Git\examples directory in Windows Explorer you should see something like this:

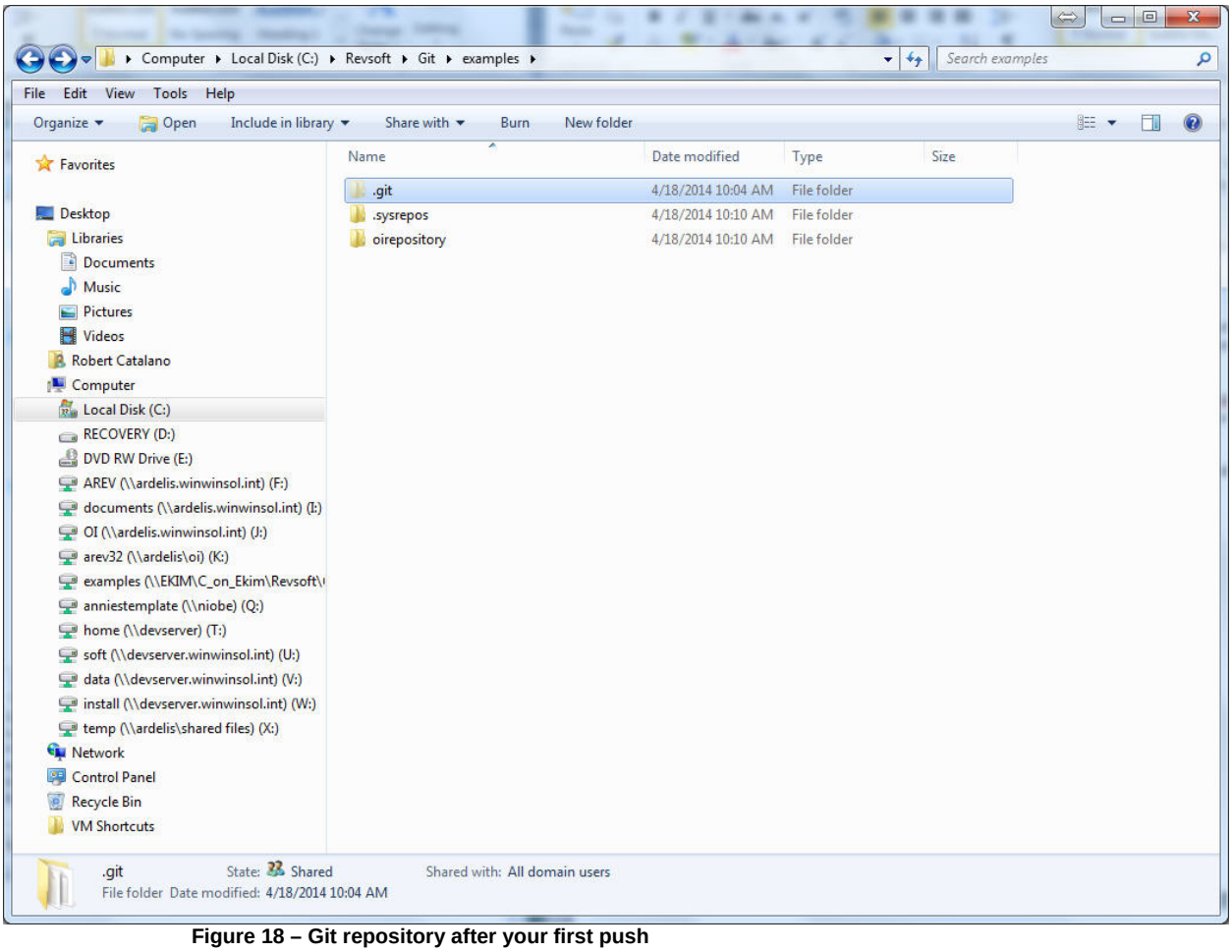

Figure 18 – Git repository after your first push

Two new directories have been created.

The **.sysrepos** directory contains partial OpenInsight repository records. These are required when pushing and pulling some types of OpenInsight repository items such as external files.

The **oirepository** directory contains the source for the repository types that you pushed to Git. The structure of this directory is a mirror of your OpenInsight repository structure.

You can browse this directory and view your source code outside of OpenInsight.

Note: Git is now monitoring these files. If you make changes to these files then Git knows about it and tracks that as a change.

Committing changes to the Git repository

Now that you have pushed your source code out to Git, the next step is to "commit" the ***changes*** to Git. Git maintains a database of the changes to files every time you commit the changes. Therefore it is important to commit to Git on a regular basis, say daily or at the end of your coding session. There is no overhead to committing often. In fact the more often you commit, the better.

To commit your changes, you can either right-click in the C:\Revsoft\Git\examples directory and select GitEx Commit or from the Application Manager, Application Tools, Git, Git Commit.

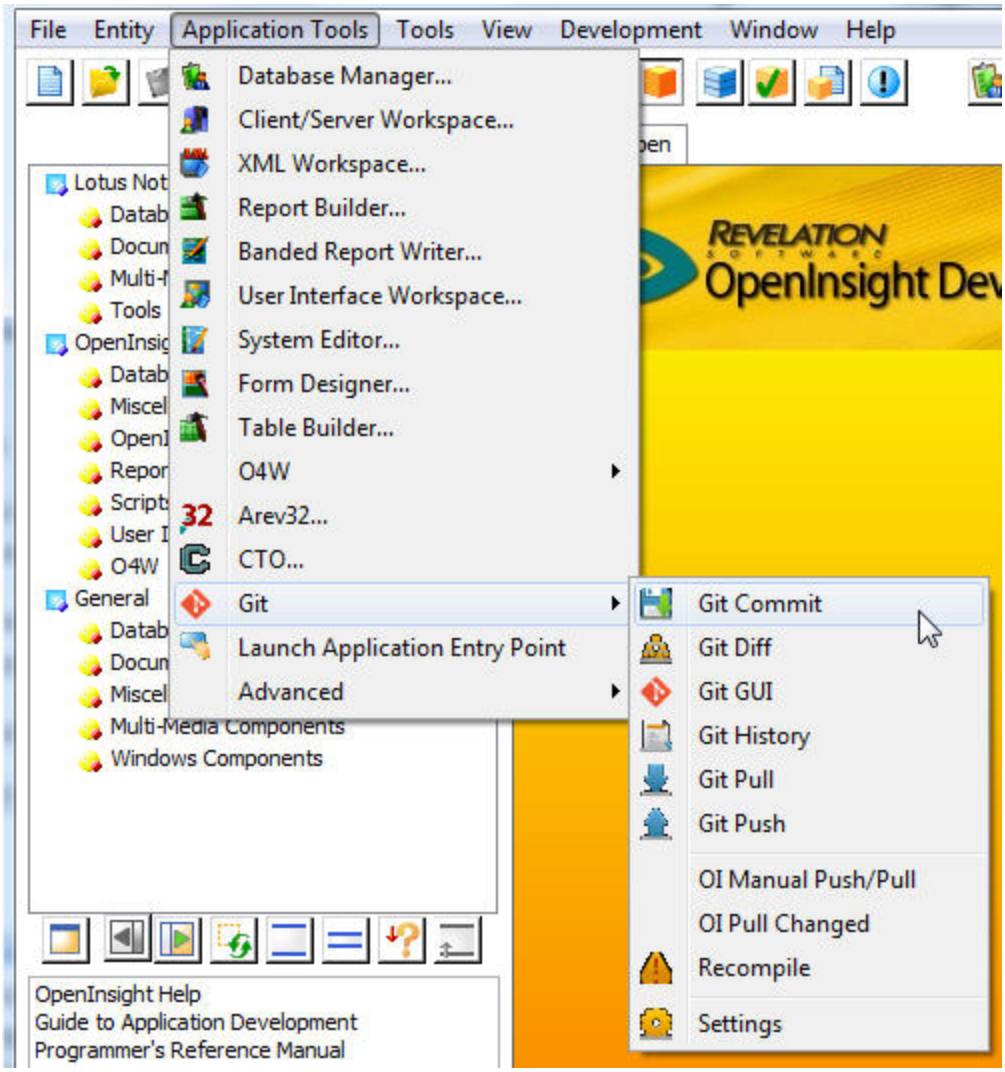

Figure 19 – Committing changes to Git

The Git Extensions commit window will display.

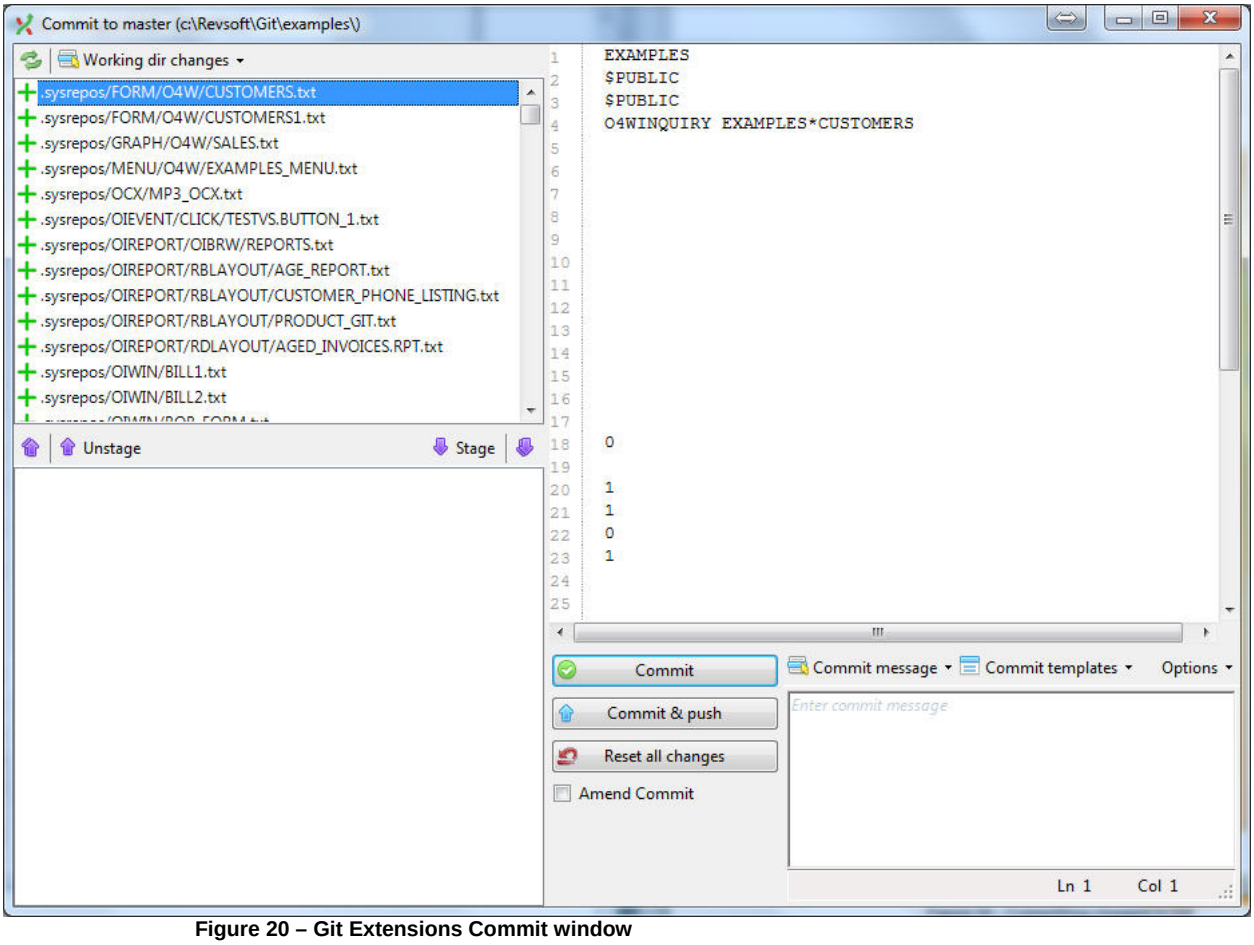

Figure 20 – Git Extensions Commit window

Most Git clients will have a similar commit window. The upper left is your *Working Directory* and contains a list of all the files that have changed since your last commit. As you can see, each of these has the green plus icon which means that these are new files. Changes will have a pencil icon and deletes will have a red minus sign .

The upper right displays the contents of the file you selected in your Working Directory.

You must stage your changes from the Working Directory to the Staging Area in the lower left before you can commit.

Only files in the Staging Area will be committed. To move files from the Working Directory to the Staging Area, use the Stage buttons.

Move all your files to the Staging Area.

Before you commit you must add a comment. This comment is used to describe the purpose of this commit. You can type as much or as little as you like. The commit process will automatically add the date, time and your Git user details to the commit so there is no need to add these details.

Type the following commit comment:

My first commit

Now click the Commit button to commit these files to Git. Committing files is like taking a snap shot of the file at a point in time. You will be able to revert back to or look at these files as they were when you committed them.

Your commit window should look like this:

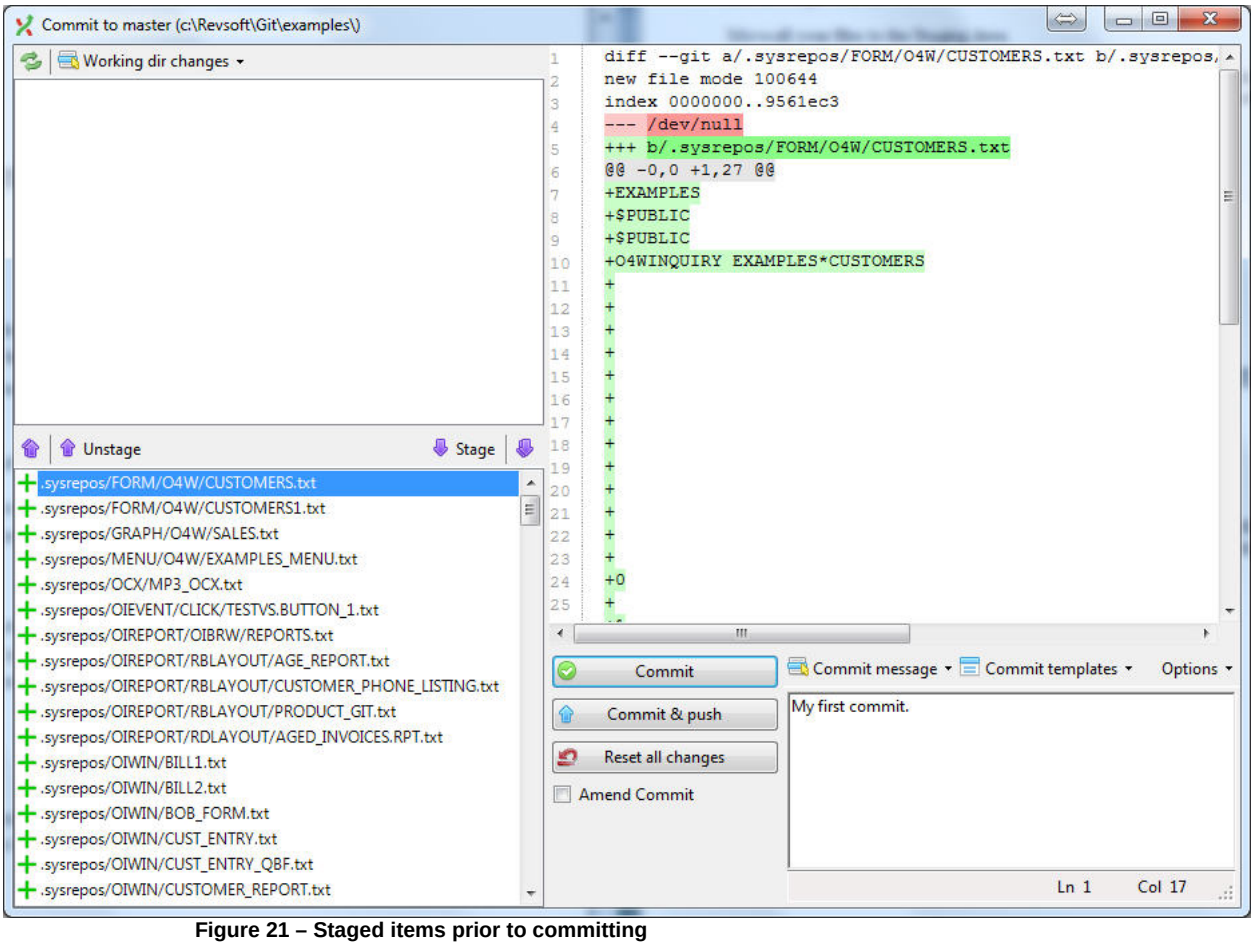

Figure 21 – Staged items prior to committing

When you click the Commit button a process window will appear. Your process window should look like this once it's completed.

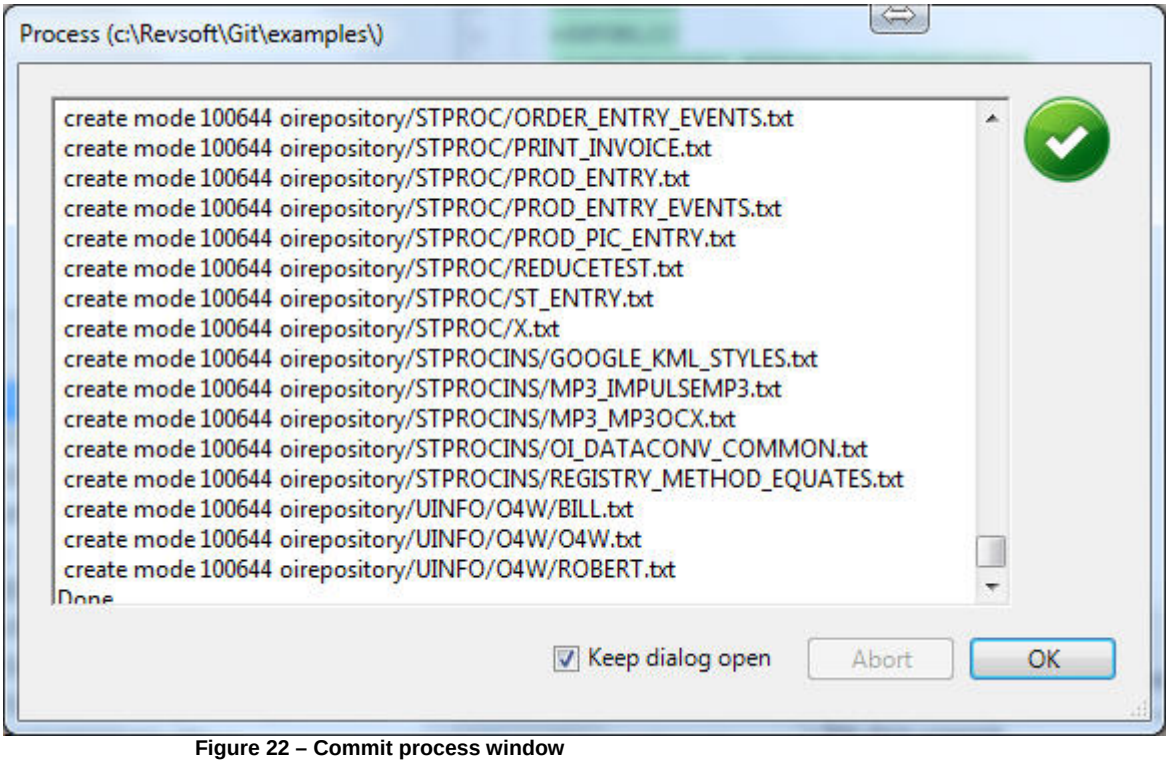

Figure 22 – Commit process window

Moving changes between multiple Git repositories

In a multi-developer environment, you would then send your changes to a remote central Git repository and receive any new changes down to your local Git repository. For the purpose of demonstrating the OpenInsight push/pull process, we are going to simulate receiving changes from another copy of OpenInsight. In the example below, the same Git environment has been set up on another workstation which we will Remote Desktop into. The system we have been working on is called Charisma and the system on the Remote Desktop is called Ekim. The flow of events are as follows:

- Create a new program on workstation Ekim in the EXAMPLES application.
- Commit the new changes to Ekim's local Git repository.
- From the Charisma workstation in the EXAMPLES application pull the Git repository changes from Ekim into the Charisma's local Git repository.
- Pull the changes from Charisma's local Git repository into the EXAMPLES application on Charisma.

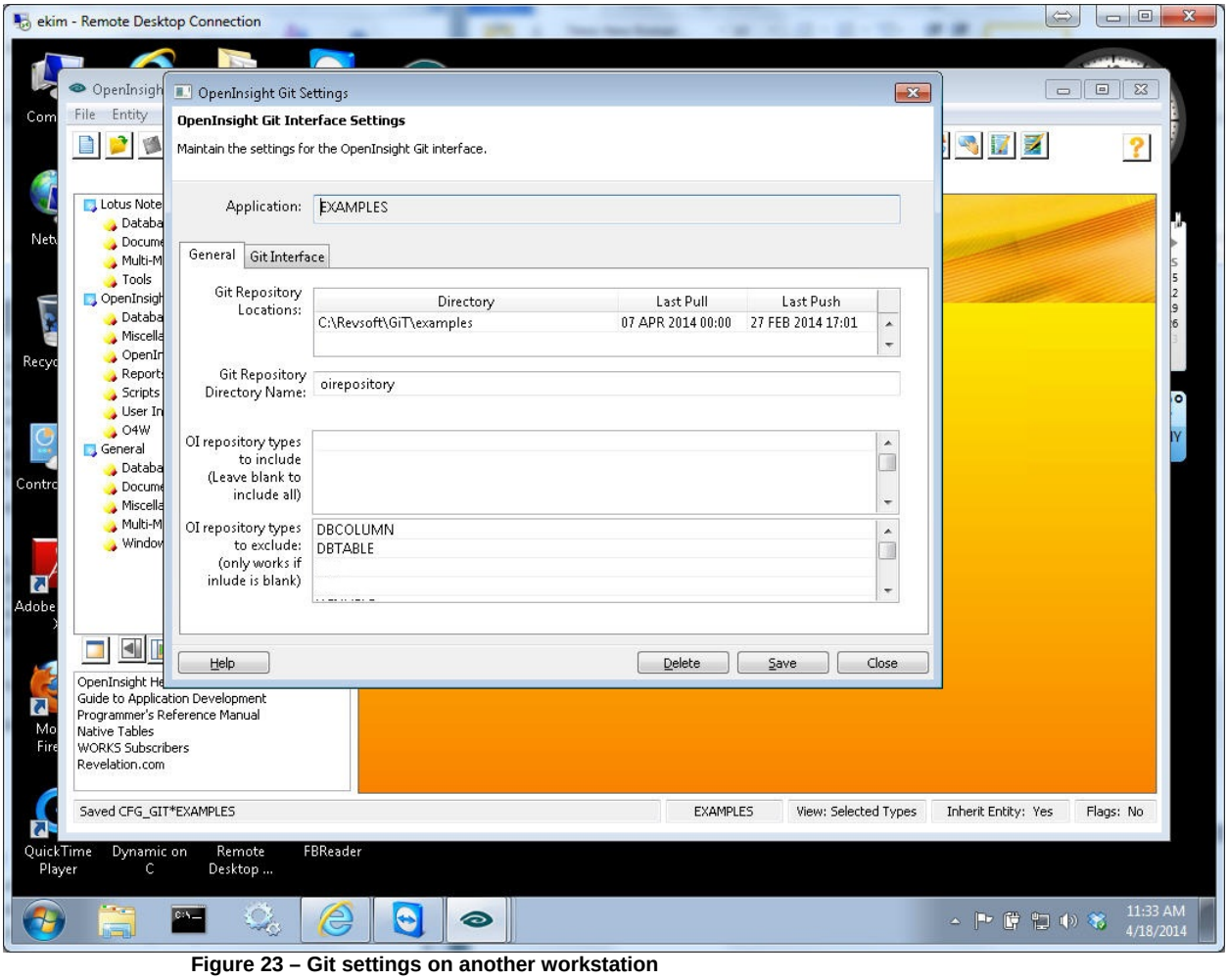

Figure 23 – Git settings on another workstation

Using the System Editor++, we will create a new stored procedure called A_NEW_PROC on Ekim.

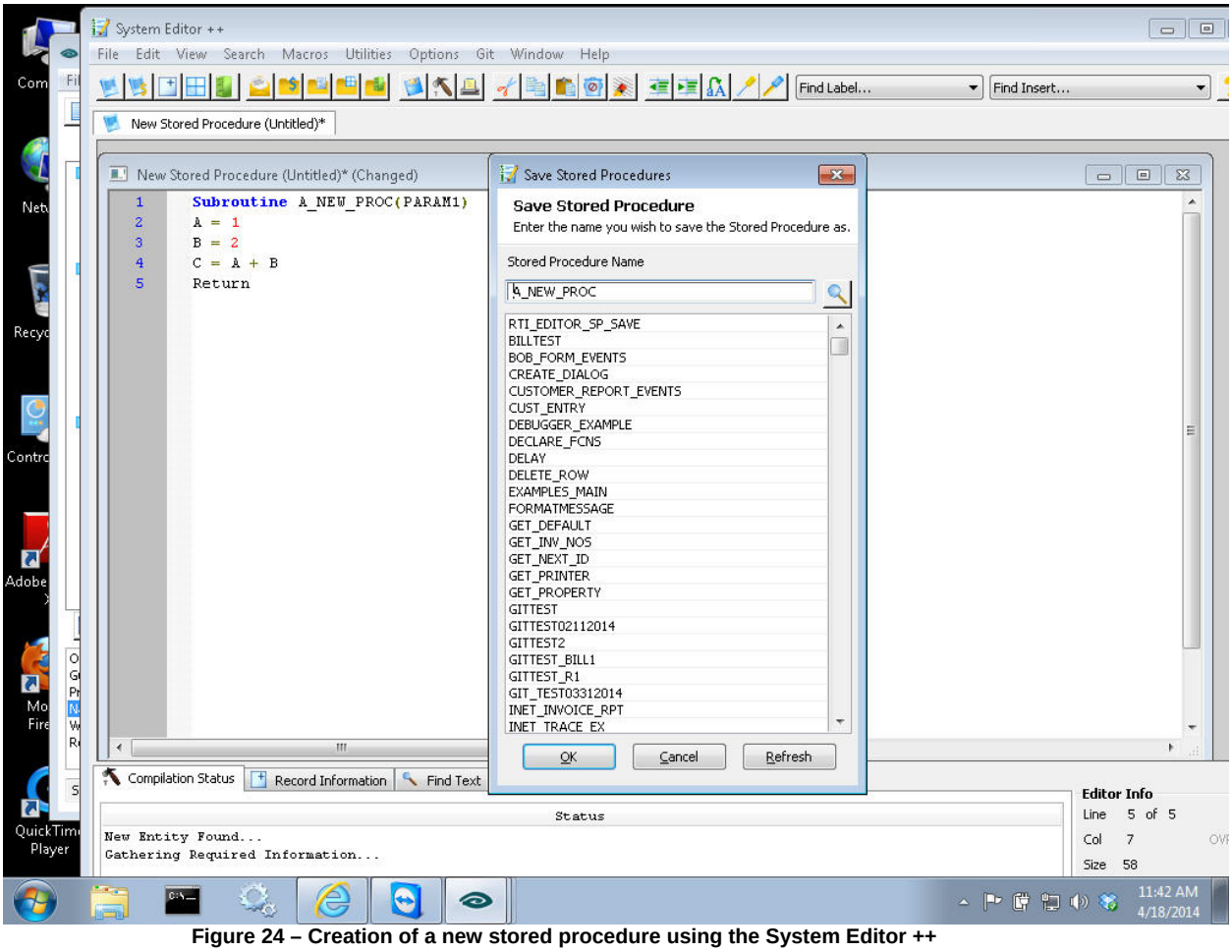

Figure 24 – Creation of a new stored procedure using the System Editor ++

From the System Editor ++ menu, select the Git menu, Git Commit. OpenInsight has automatically pushed the newly created stored procedure into the local Git repository on Ekim.

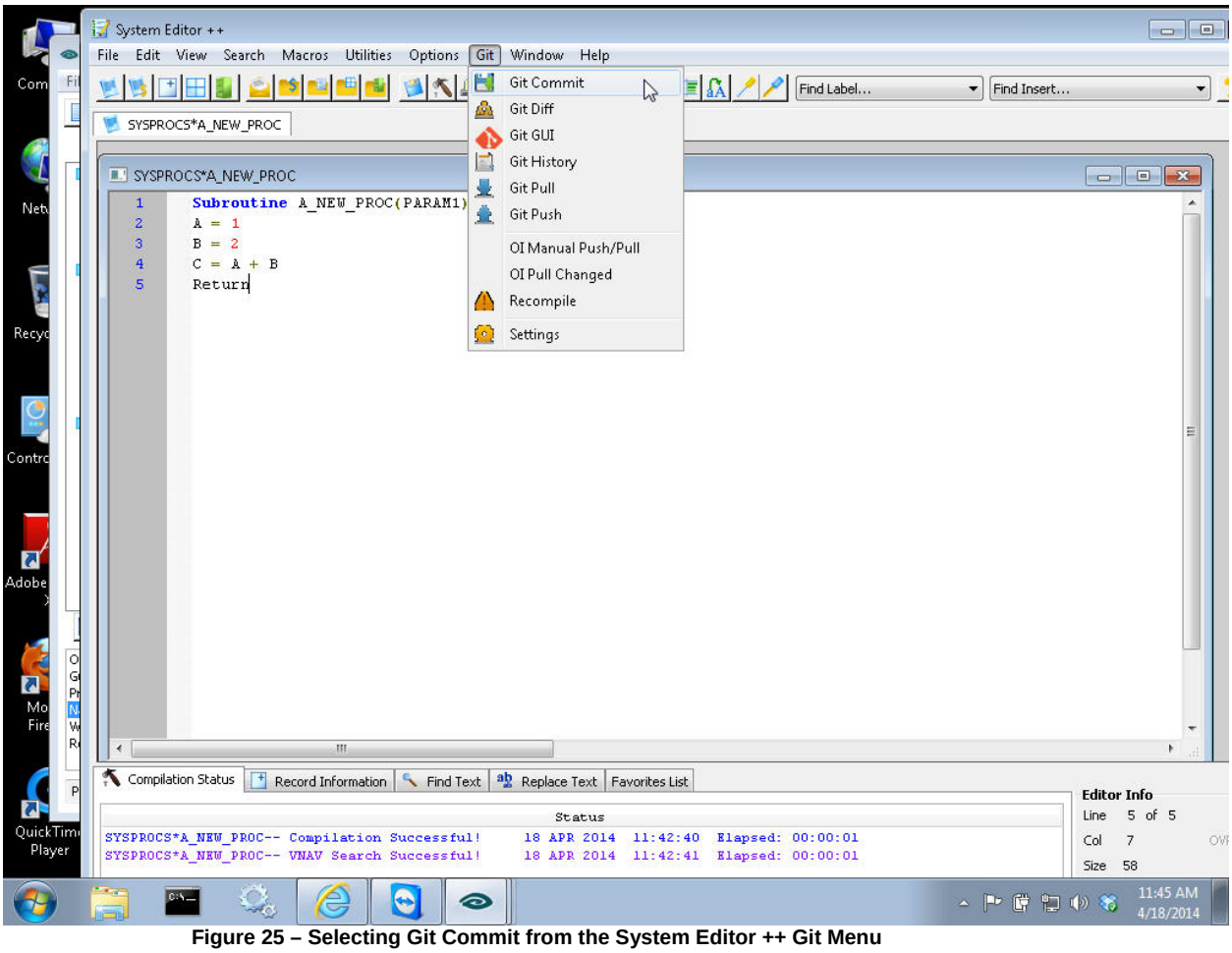

Figure 25 – Selecting Git Commit from the System Editor ++ Git Menu

The Commit window will appear again and look like this.

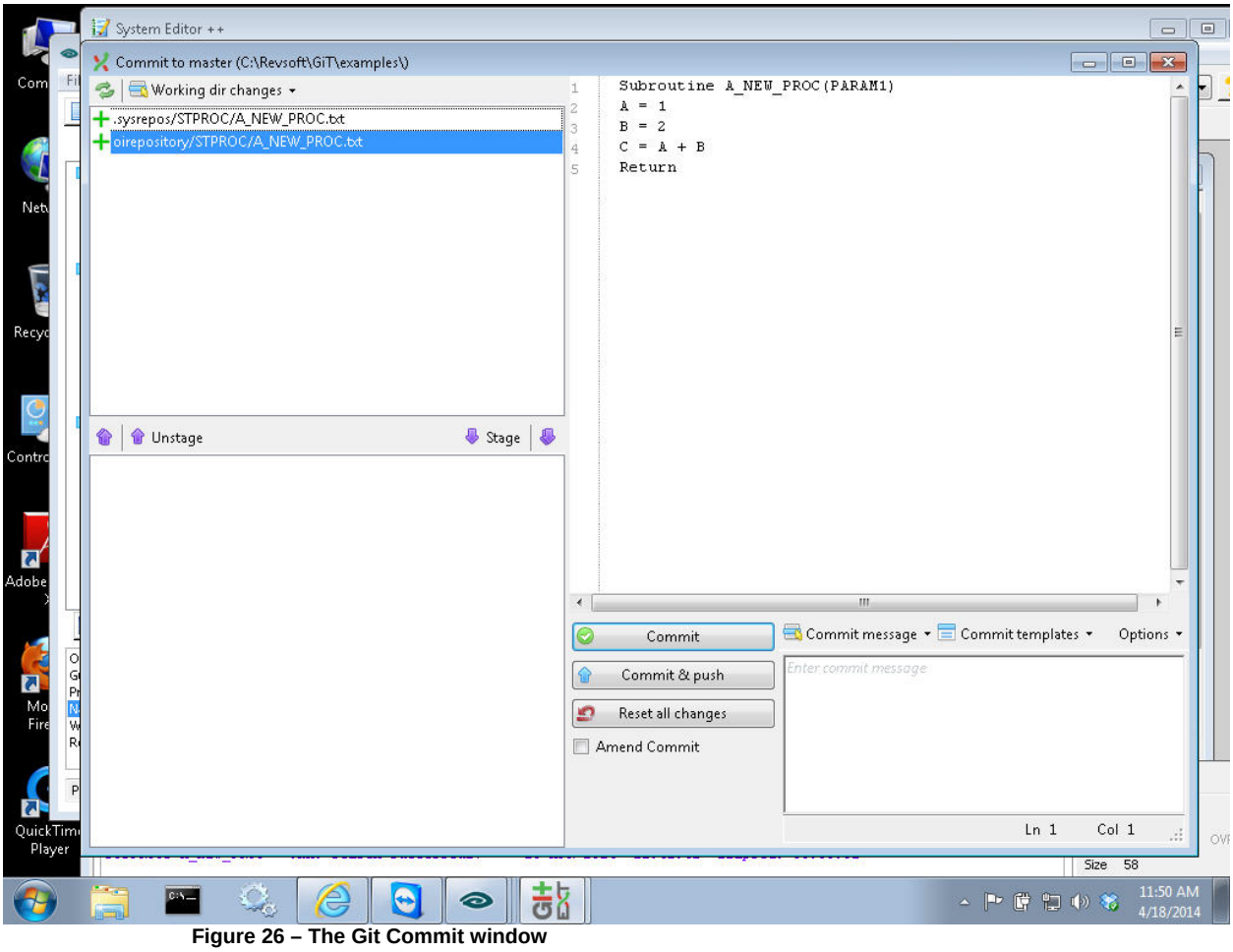

Figure 26 – The Git Commit window

As you can see, Git knows that there is a new file called A_NEW_PROC.txt. We will now stage the items and commit to the local repository on Ekim. After clicking the Commit button these changes are now committed to the Git repository. The next step from the Charisma workstation is to "pull" these changes from the local Git repository on Ekim into the local Git repository on Charisma.

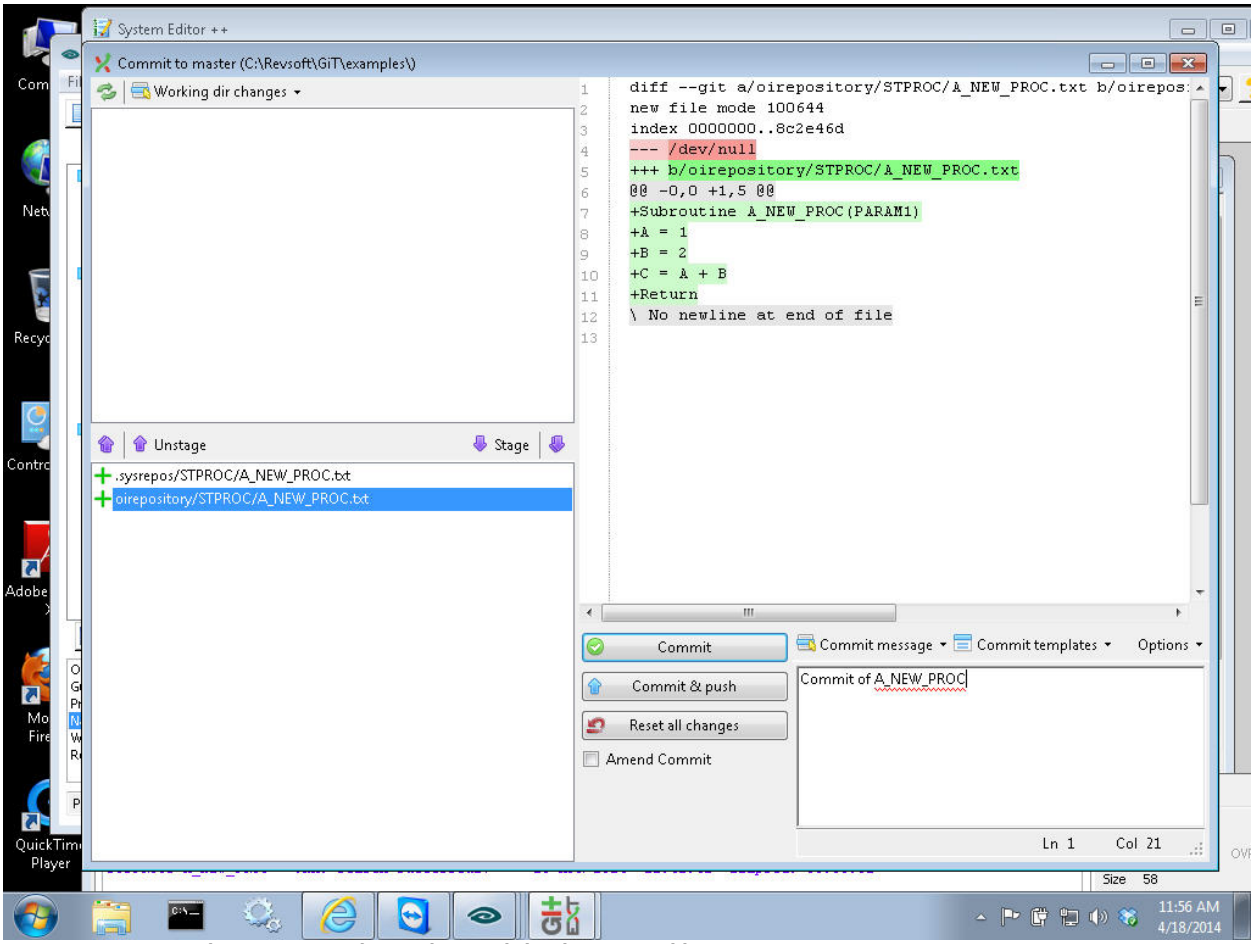

Figure 27 – Staging and committing items on Ekim

Pulling changes from one Git repository into another Git repository

To pull changes from the Git repository on Ekim into the Git repository on Charisma, choose Git from the Application Manager on Charisma, then Git Pull.

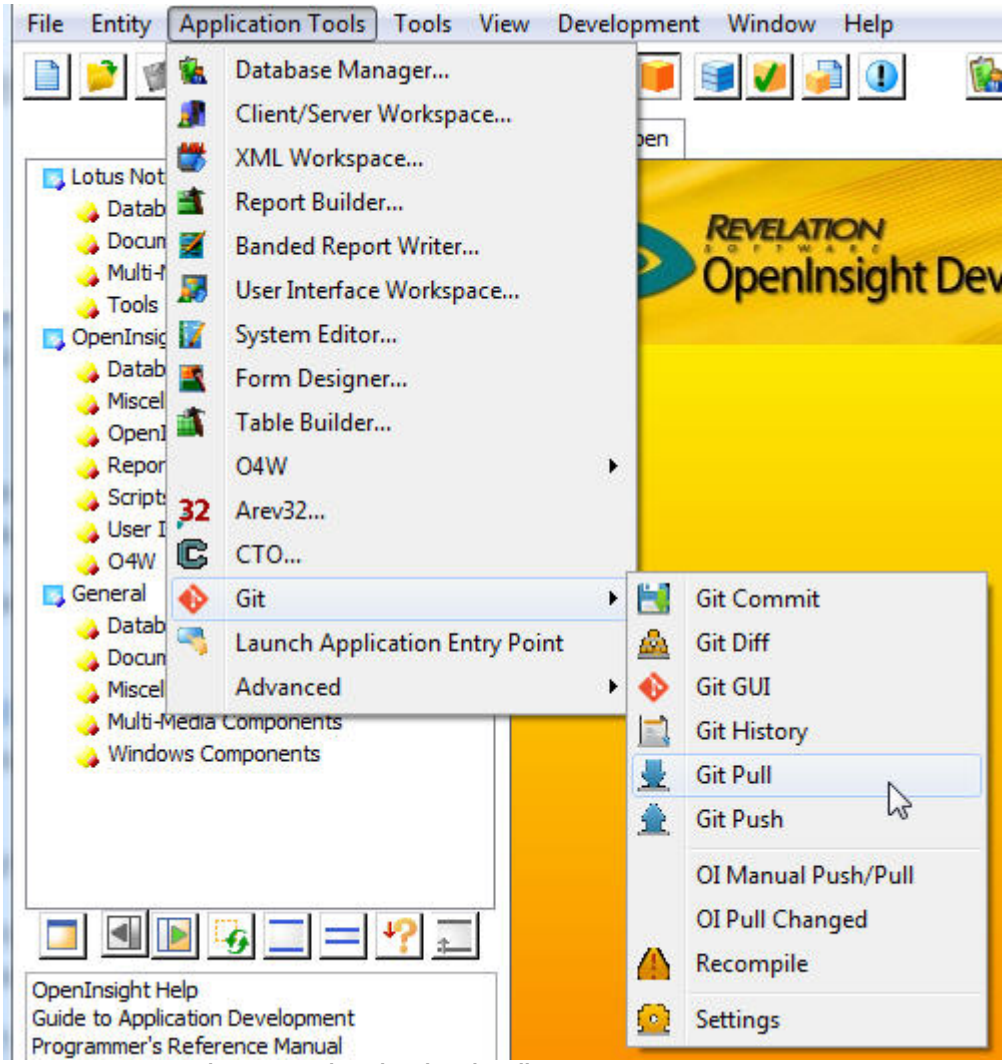

Figure 28 – Choosing the Git Pull menu

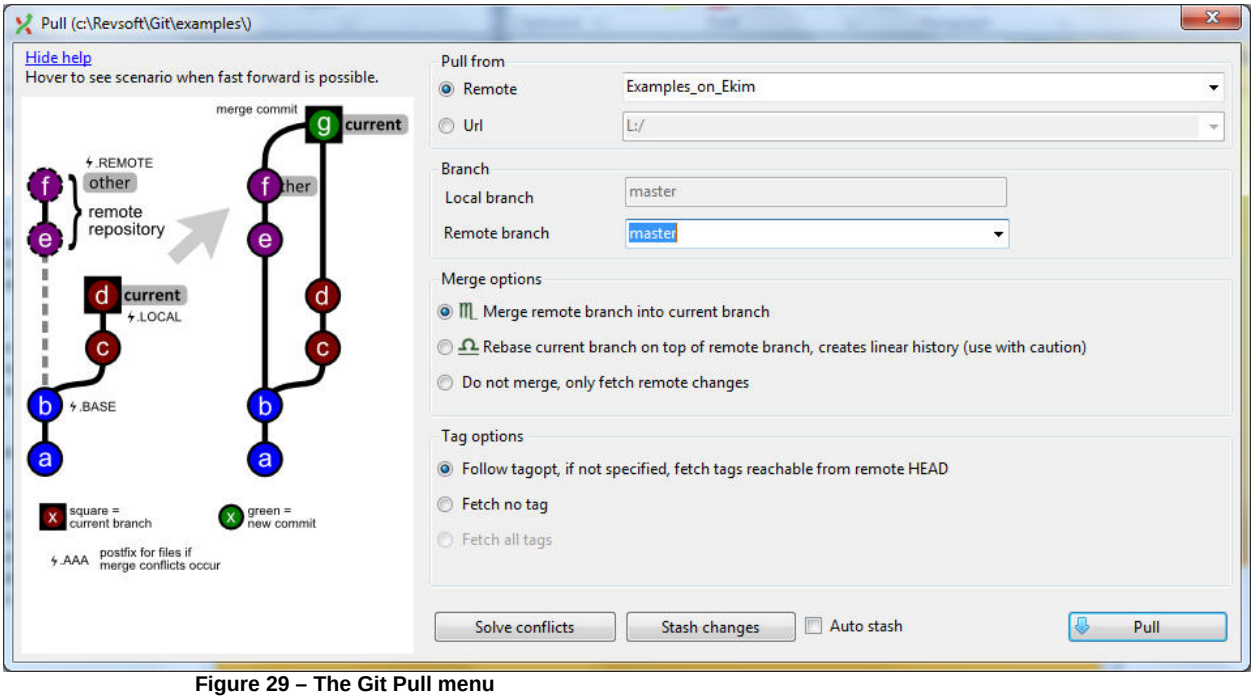

Figure 29 – The Git Pull menu

Note: Pull from: Example_on_Ekim is mapped to \\Ekim\C$\Revsoft\Git\examples on our charisma workstation. Click the Pull button to merge the remote branch (Ekim repository) into the current branch (Charisma repository).

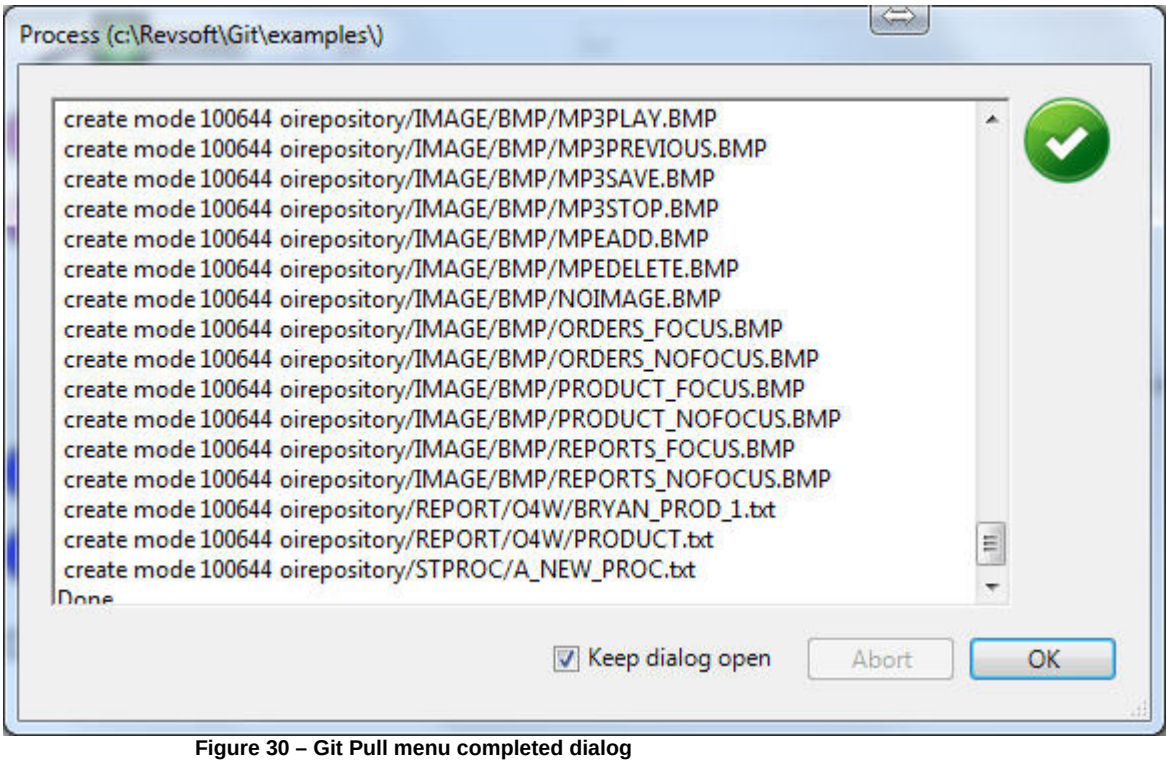

Figure 30 – Git Pull menu completed dialog

Pulling changes from Git into OpenInsight

To pull changes from the Git Repository on Charisma into the OpenInsight on Charisma choose Git, OI Manual Push/Pull.

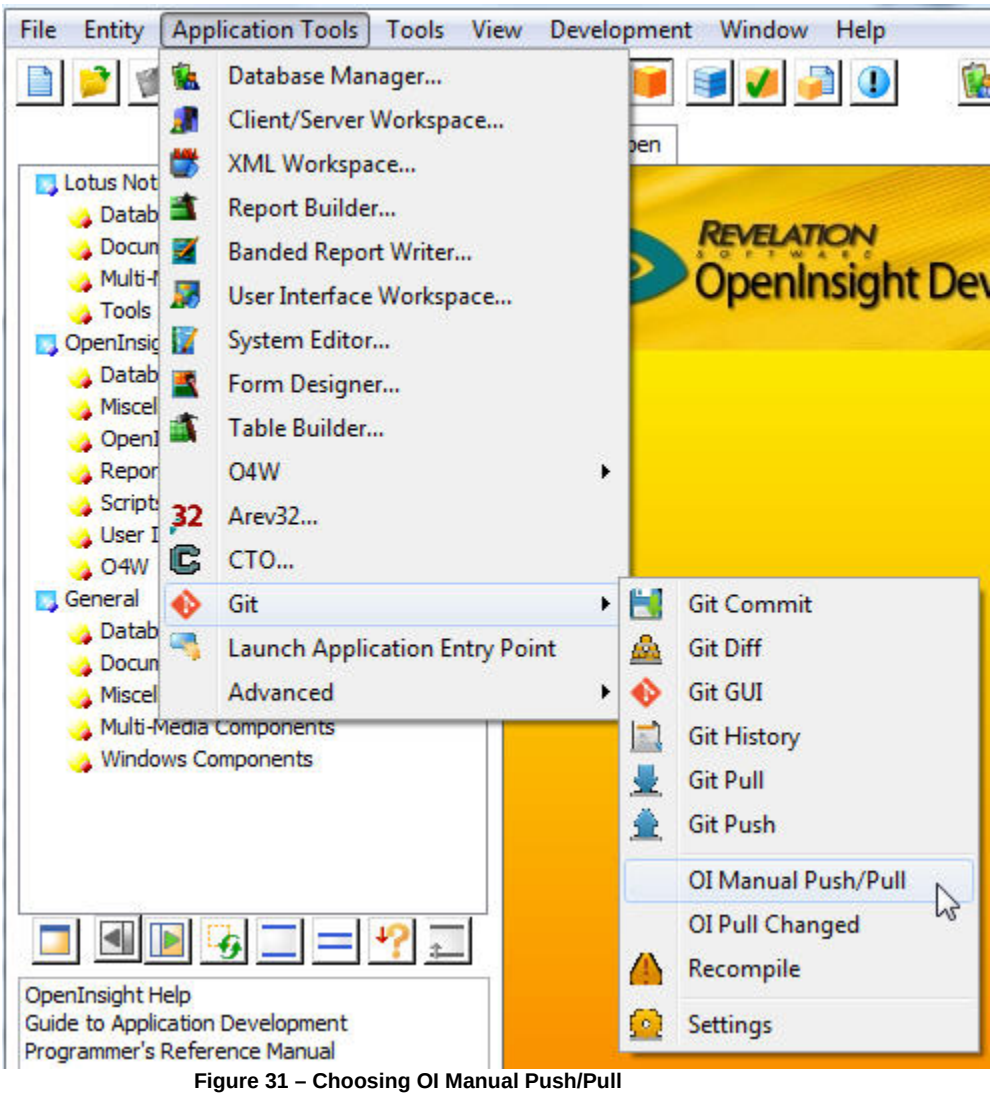

Figure 31 – Choosing OI Manual Push/Pull

1. At Operation select “Pull”
2. At Selection Type select “Changed”

When you select “Changed” OpenInsight Git will compare your OpenInsight source with the source in Git and only select the items that don’t match.

*Note: **THIS IS IMPORTANT**, the “**changed**” option only knows that the item in Git and the item in OpenInsight are not the same. It does not know which one is the latest version. However, it is reasonable to assume that if the Git version and OpenInsight version are different, then the Git version must have been pulled in from a remote server. You can use the Compare button to compare the OpenInsight and Git version.*

Your OpenInsight Git window should look something like this:

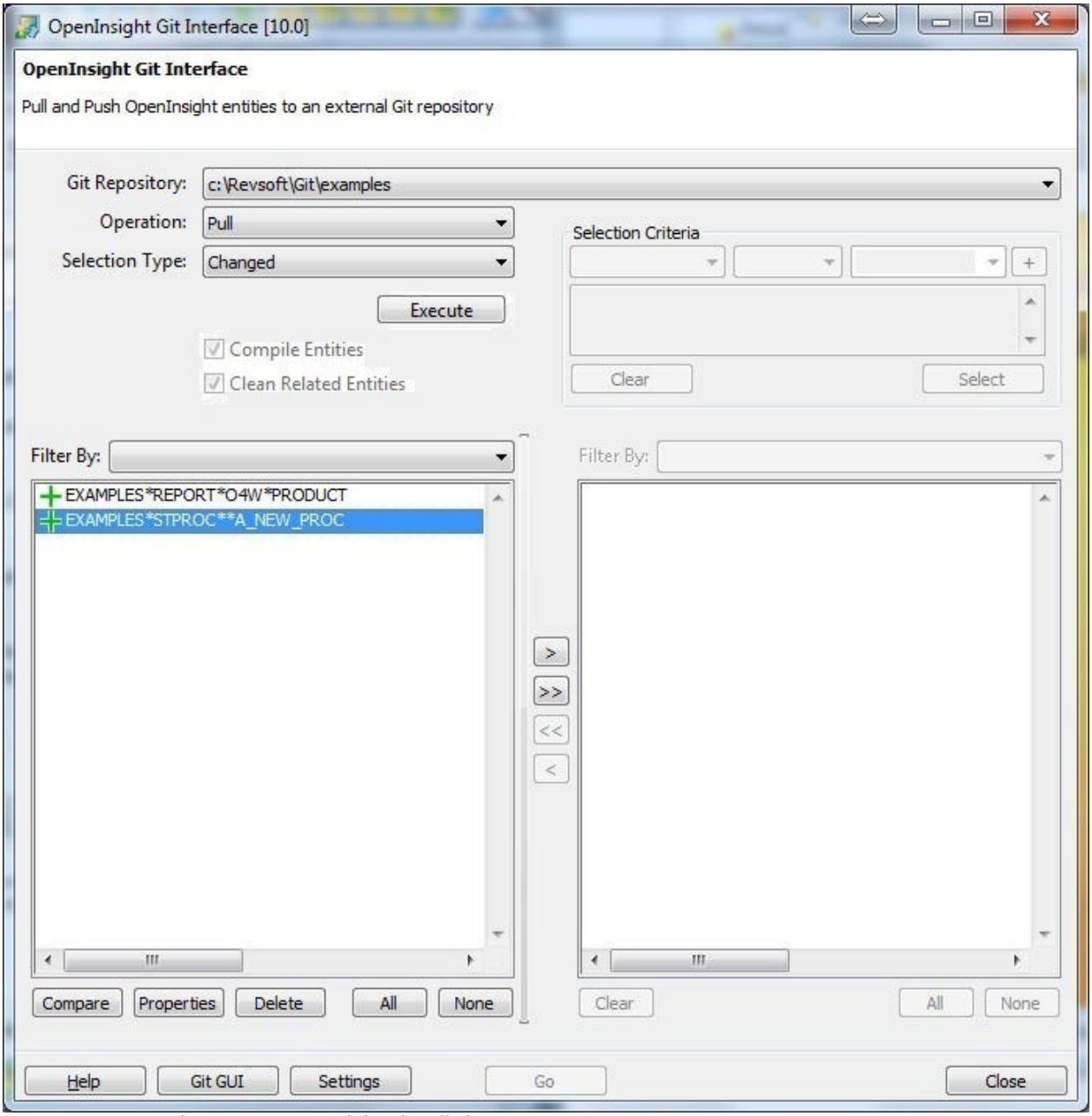

Figure 32 – OpenInsight Git pull changes

Just as in the push operation above, you need to select the items you wish to pull into OpenInsight. Move the items you wish to pull into the box on the right.

Note that the new program, A_NEW_PROC.txt, has a green plus icon next to it indicating that this procedure exists in your Git repository but not in your OpenInsight repository. Items that have a pencil icon exist in both Git and OpenInsight but they are different.

You can compare the OpenInsight and Git versions by selecting the A_NEW_PROC procedure in the list on the left then clicking on the Compare button. The compare window will appear.

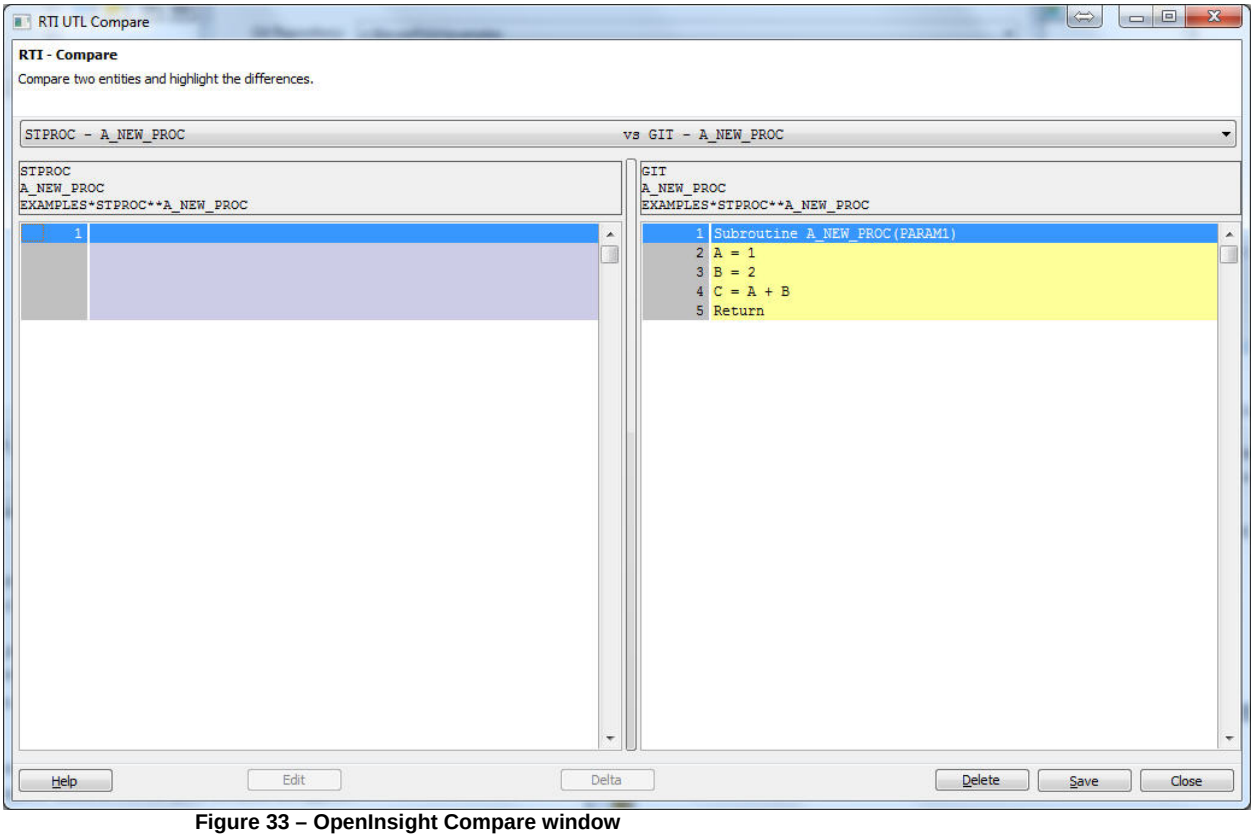

Figure 33 – OpenInsight Compare window

A copy of the OpenInsight source code is on the left and a copy of the Git source code is on the right.

A yellow highlight indicates that something has changed on the line.

A green highlight indicates that this is a new line.

A red highlight indicates that a line has been deleted.

If you are unsure if your Git or OpenInsight version is the latest version, you can use this window to compare each of them.

Click Close to close the Compare window.

After moving both procedures to the box on the right, your window should look something like this.

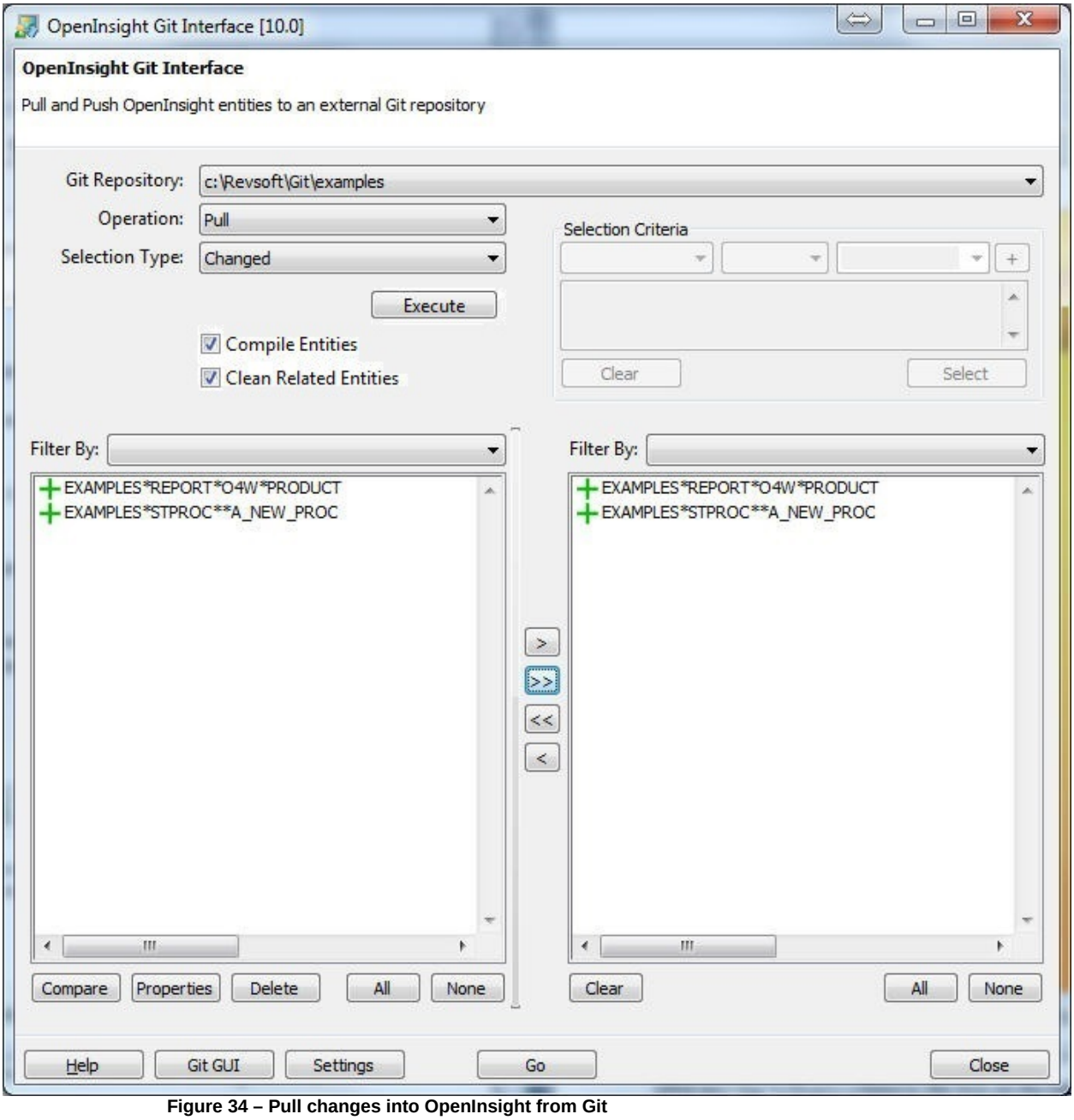

Figure 34 – Pull changes into OpenInsight from Git

Check Compile Entities, Clean releated entities and Click the Go button and OpenInsight Git will pull these changes into OpenInsight and compile and clean the entities.

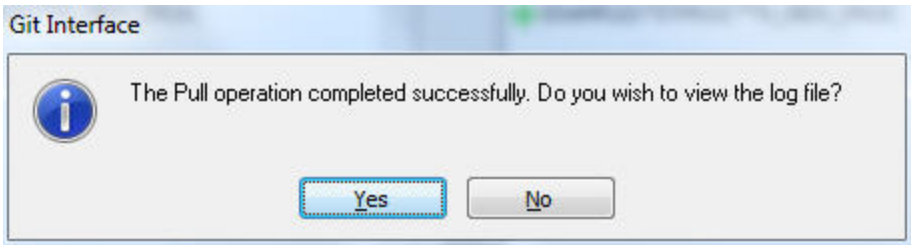

Figure 35 – Pull completion dialog

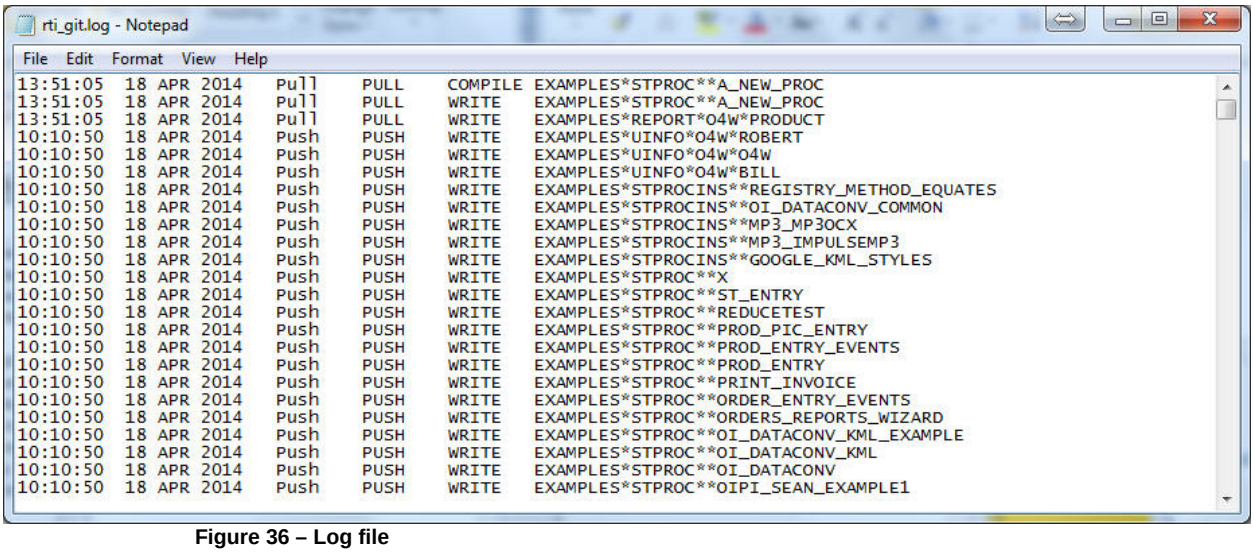

Figure 36 – Log file

*Note: **THIS IS IMPORTANT** - pulling changes into OpenInsight from Git will completely replace the source code in OpenInsight.*

Note: When you pull items in from Git, if the item can be compiled, like a stored procedure or an OpenInsight window, then OpenInsight Git will compile the item to ensure that the source and object versions of the item are in sync.

If you pull an insert in from Git, then all procedures that use that insert will be recompiled.

You can now close the OpenInsight Git window. Open your editor and edit the A_NEW_PROC procedure. You will see the that new stored procedure we created on Ekim has been added as a new stored procedure on Charisma.

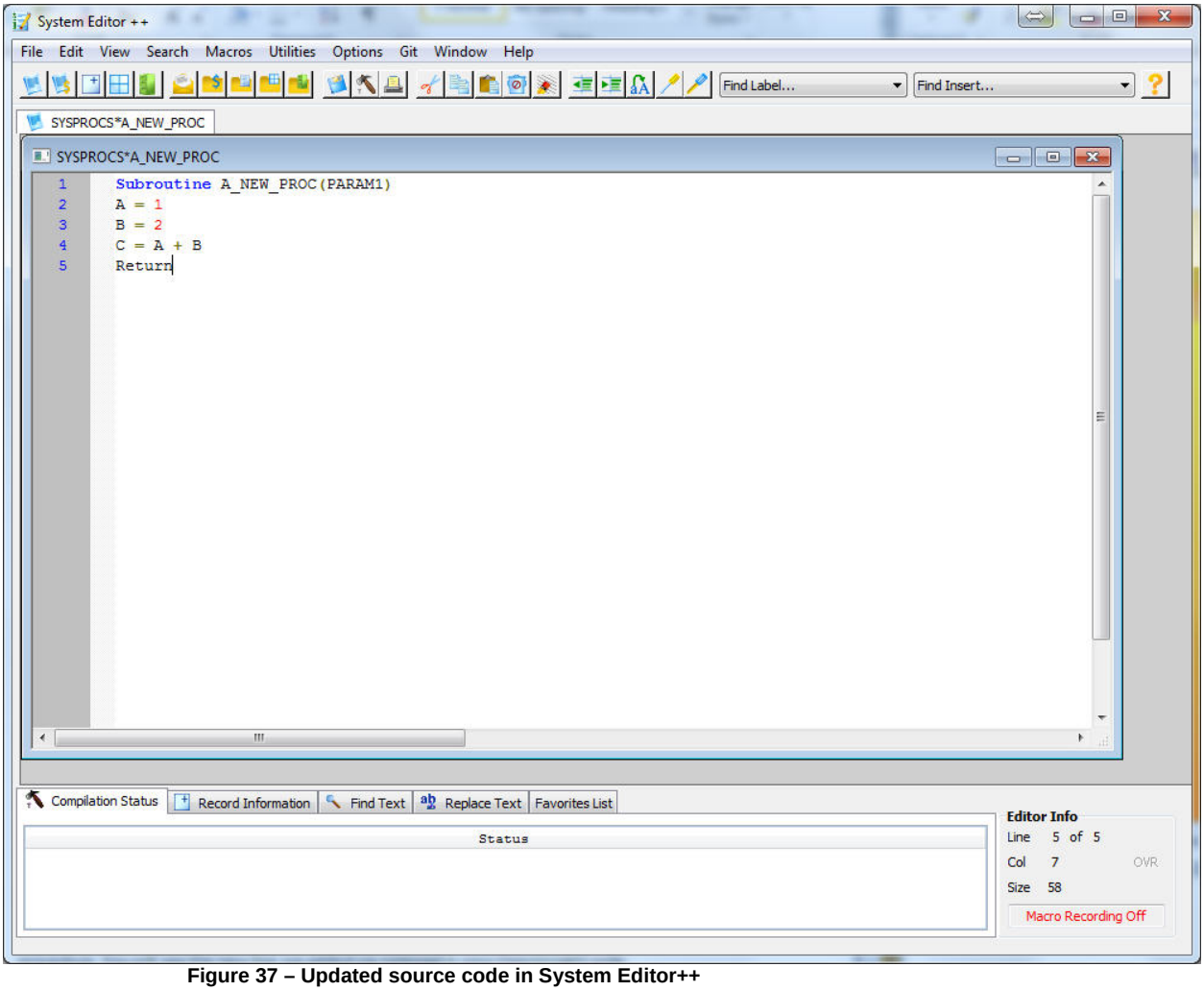

Figure 37 – Updated source code in System Editor++

OpenInsight Git automatic push to Git repository

Any changes you save in OpenInsight will automatically be pushed out to Git. For example, when you save a stored procedure in the System Editor++, the updated source code will be automatically pushed to your Git repository.

This automatic push will occur for all repository entities when using their default editing tool. It will not work if you update an entity by bypassing the repository function. For example, if you edit a procedure in the System Editor ++ by opening a record in the SYSPROCS table, making changes then saving the changes. Tthis method bypasses the REPOSITORY function and also the automatic push to Git.

So let's see how this automatic push works. Open the System Editor ++ and open the procedure CUST_ENTRY

Insert a new line at line 2 like this:

* 18 April 2014 - Added this line in OpenInsight

Then either Save or Save and Compile the change. You will not see any difference in OpenInsight, but your changes have been automatically pushed to Git. Automatically pushing the changes for you makes working with Git much easier.

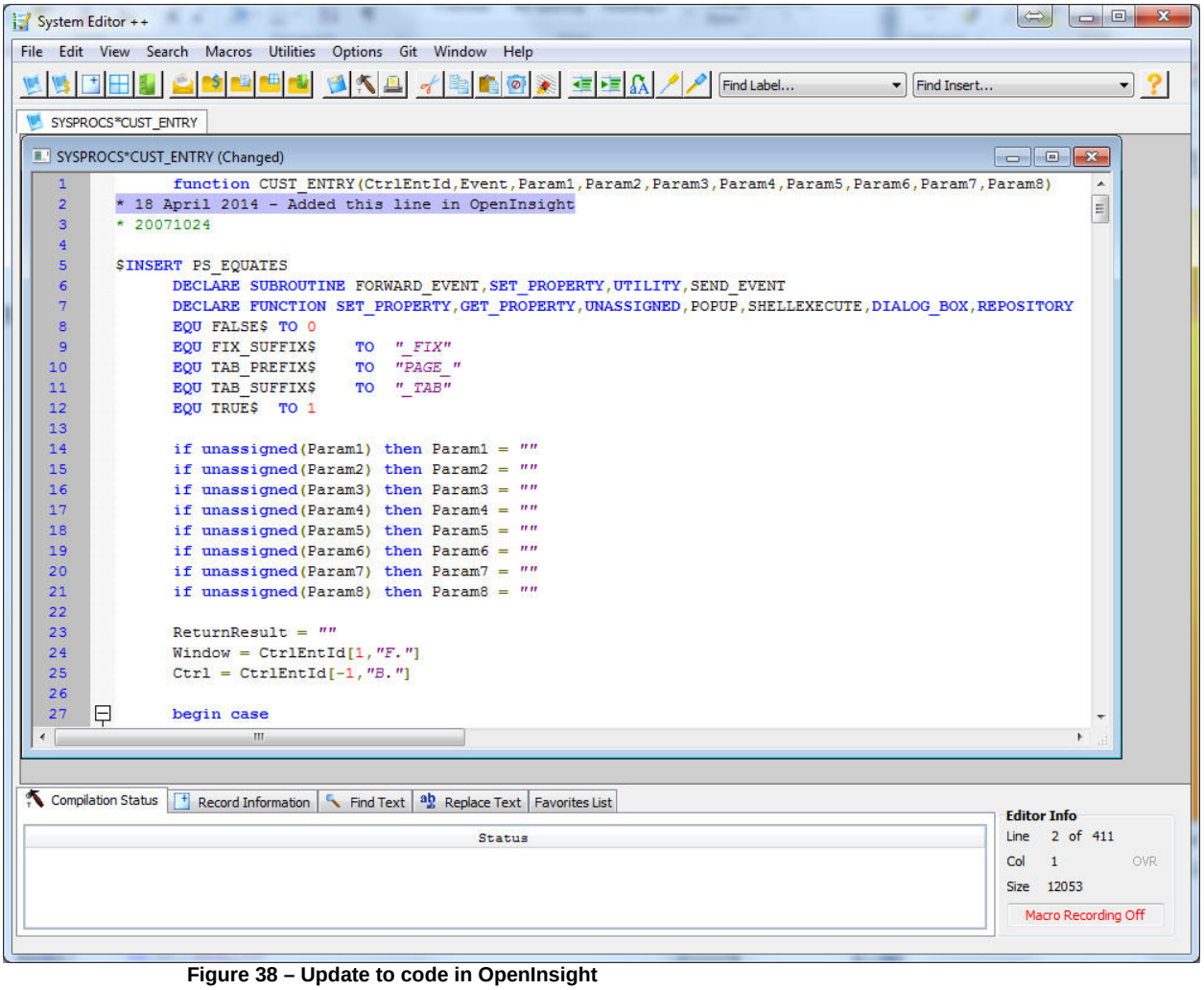

Figure 38 – Update to code in OpenInsight

Now go back to Windows Explorer and navigate to C:\Revsoft\Git\examples\oirepository\STPROC and edit the CUST_ENTRY.txt file in Notepad. You will see your newly added line near the top.

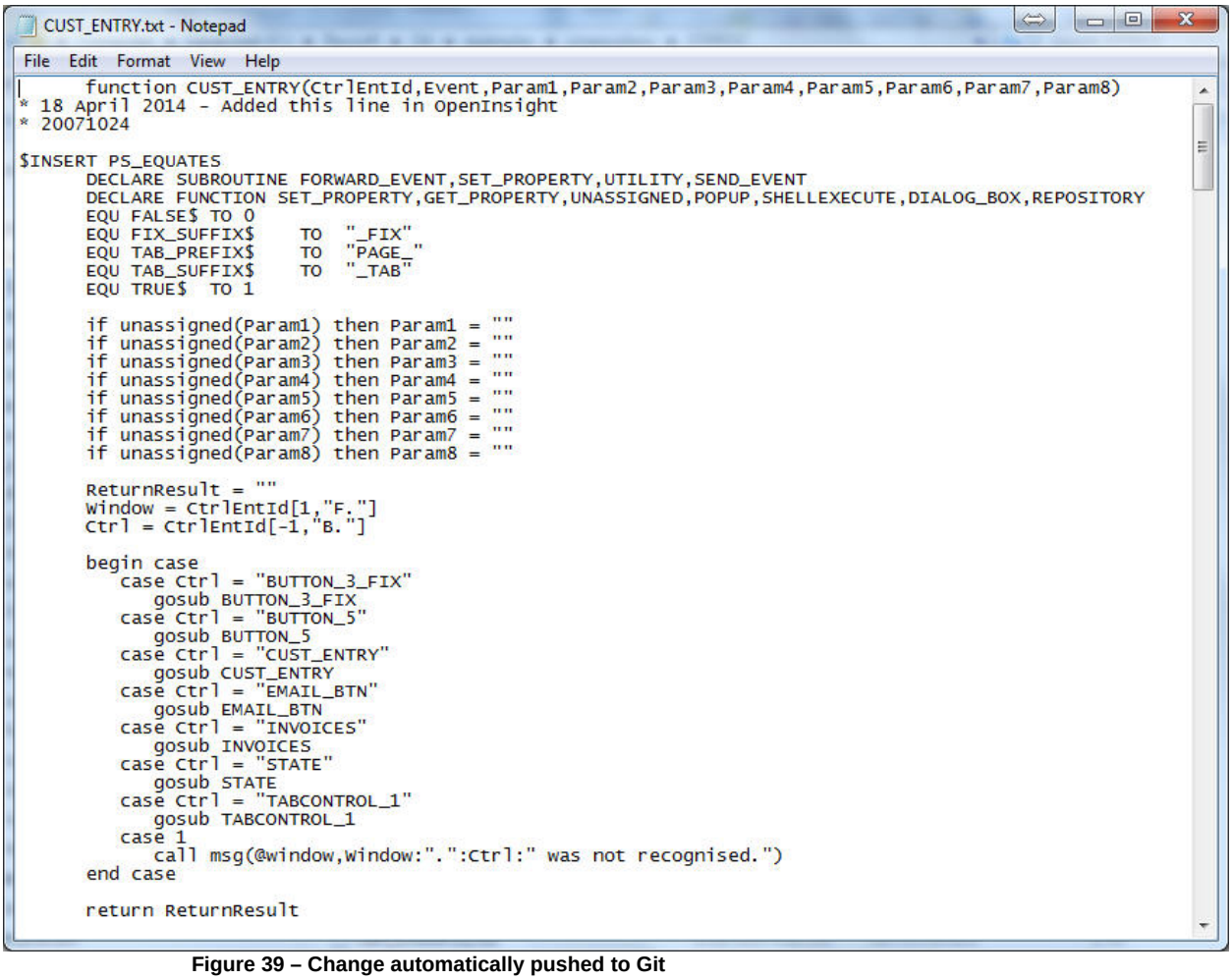

Figure 39 – Change automatically pushed to Git

It does not matter how many times you save changes in OpenInsight, Git will only remember the source code as of the last save you made in OpenInsight. To get Git to “snapshot” the code as at a particular time, you need to commit your changes to Git.

From the System Editor++ Git menu, select Git Commit. The commit window will look like this.

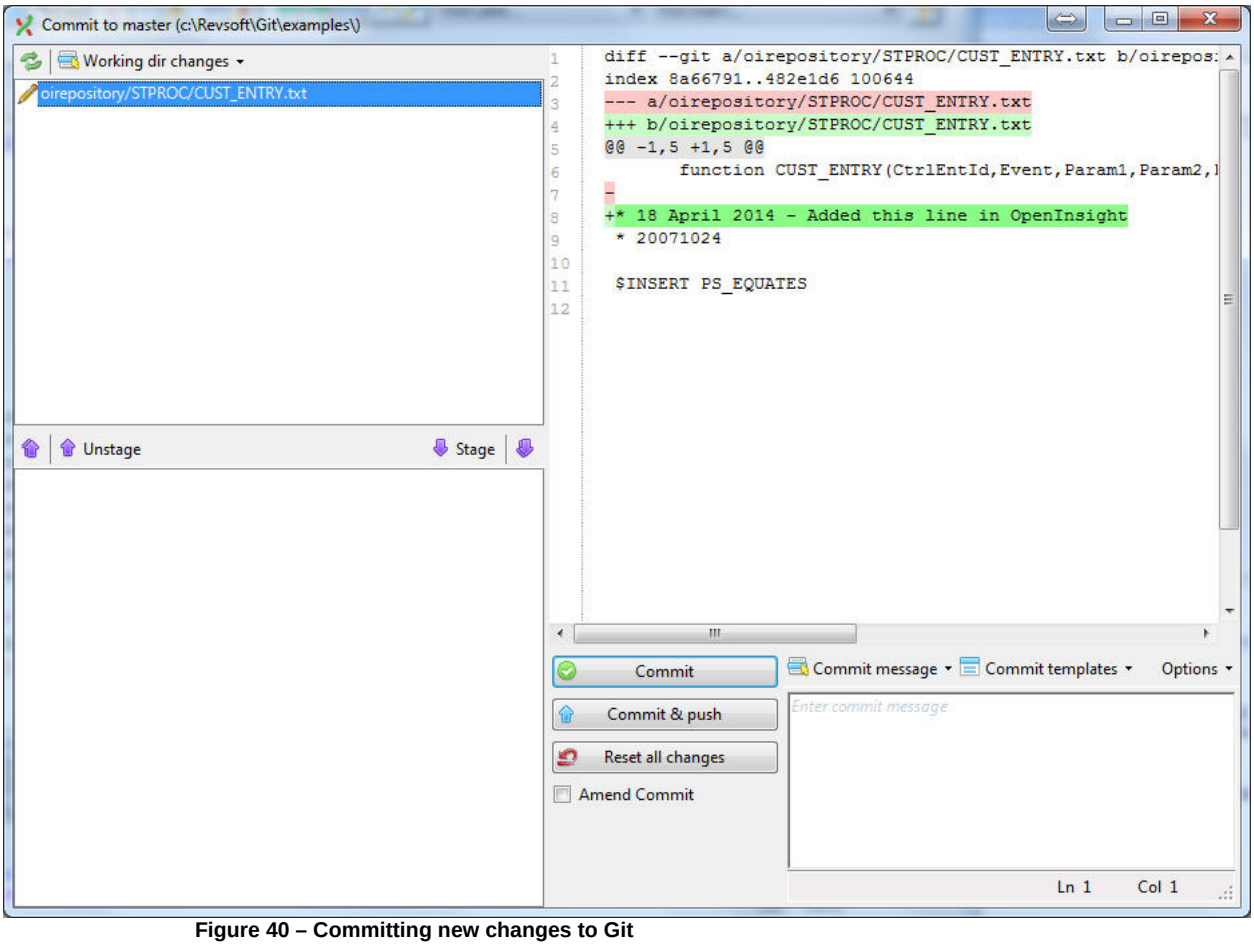

Figure 40 – Committing new changes to Git

Git knows that the CUST_ENTRY procedure has changed. Move this file into the staging area and enter a commit comment such as:

I updated this with a new comment

Then, hit the Commit button.

You now have two snapshots of the source code for CUST_ENTRY. That is, versions when:

1. You first pushed out the source code from OpenInsight and committed the changes and,
2. When you edited the program in the System Editor++ and committed the changes

To see a history of changes for this file in Git, navigate to the C:\Revsoft\Git\examples\oirepository\STPROC directory and right-click on the CUST_ENTRY.txt file.

Then, from the context menu, select Git Extensions and then, File History.

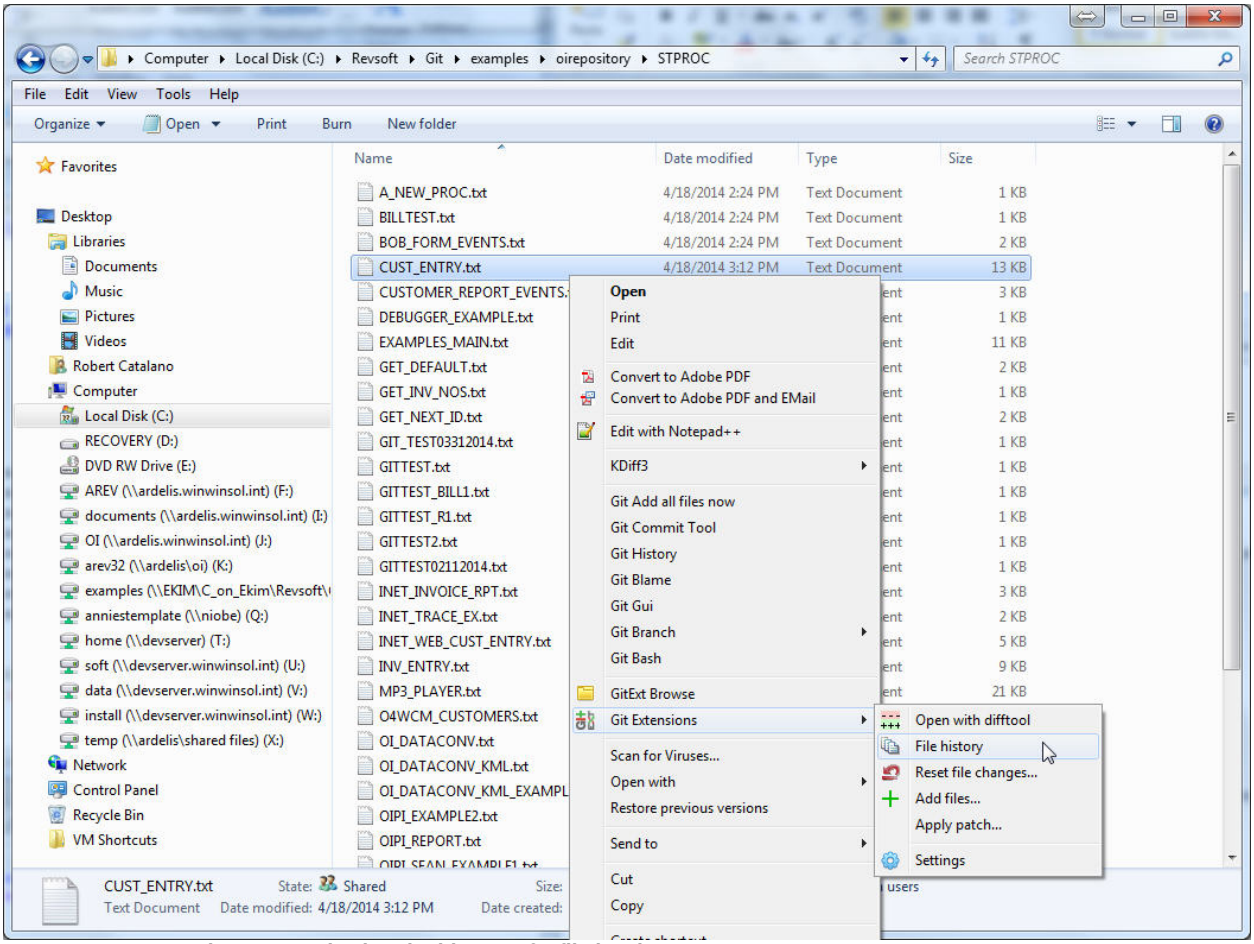

Figure 41 – Viewing the history of a file in Git

This will open the Git Extensions history window.

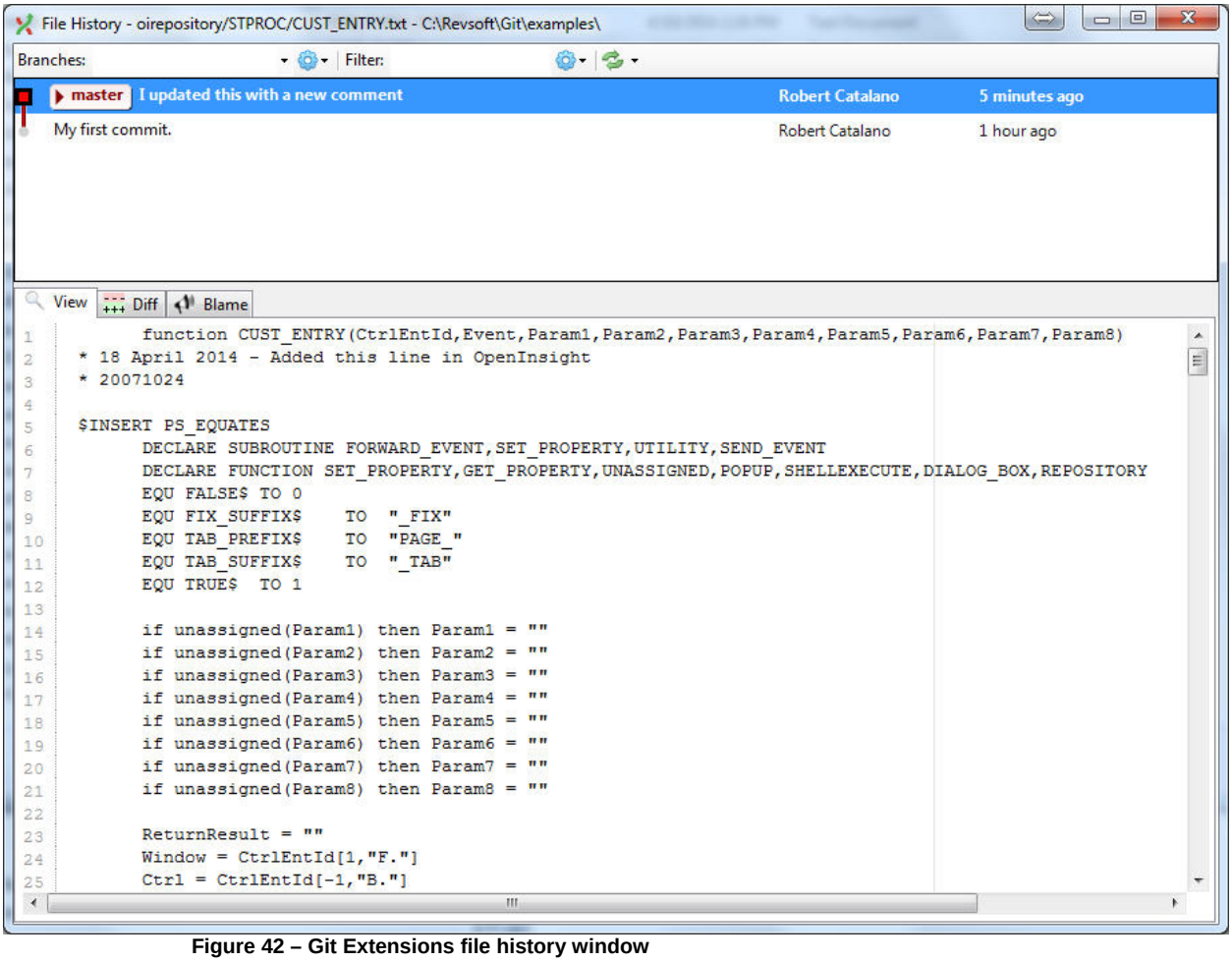

Figure 42 – Git Extensions file history window

You can see that there have been three commits for this file. You can view the source code as it was at each commit, and you can even compare the changes between each commit.

This is one of the powerful features of Git. It lets you view previous versions of a procedure and compare them to see what changed between each version, who made the change and when the change occurred.

Would you like to revert back to a previous version of a procedure, no problem. Just select the previous version you want, copy the code, and paste it into the editor in OpenInsight. Done. Also, there is no need to worry about losing the current version of the code in OpenInsight. As long as it has been committed to Git, you will always have a snapshot of the code as it was at that point in time.

Even if you delete a procedure from OpenInsight and commit the delete to Git, all is not lost. Git will always remember the snapshots of files, even after they have been deleted. Therefore, you can always get your source back. The only catch is that you must commit your changes to Git on a regular basis because Git only remembers committed files.

Remember, commit early and commit often!

The Seven Steps to Using OpenInsight Git

You should follow these simple steps when using a Git repository with OpenInsight. Depending on the Git client you use, some of these steps may be combined together, e.g. Pull and Merge. Also, the first step, Push your updates from OpenInsight, is automatically done for you in OpenInsight v10.0 and above.

1. Push your updates from OpenInsight to your local Git repository.
2. Commit your updates to your local Git repository.
3. Pull updates from remote Git repositories.
4. Merge updates pulled from remote Git repositories with your local Git repository.
5. Resolve any conflicts.
6. Push updates to remote Git repositories.
7. Pull updates from your local Git repository into OpenInsight.

Pushing updates from OpenInsight to Git

From OpenInsight v10.0 and above, OpenInsight Git is fully integrated into the OpenInsight repository. Whenever you add, modify or delete an OpenInsight entity, the update is automatically pushed to your local Git working directory. Git tracks changes, or deltas, only. If you save an entity without making any changes, Git will not recognize any changes and Git will ignore the save.

From OpenInsight v10.0 and above, updates are automatically pushed to Git. Therefore, there is usually no need to perform a manual “Push” of updates to Git. However, there are some scenarios where you must perform a manual “Push”.

1. When you first start using Git for your source code management.
2. If your local Git repository becomes corrupted.
3. If you think that something has been missed by the automatic repository processing.

To perform a manual “Push” from OpenInsight to Git either:

1. From the Application Manager, Application Tools and select the OI Manual Push/Pull operation
   or
2. Select the OI Manual Push/Pull menu option from the Git menu in the System Editor++

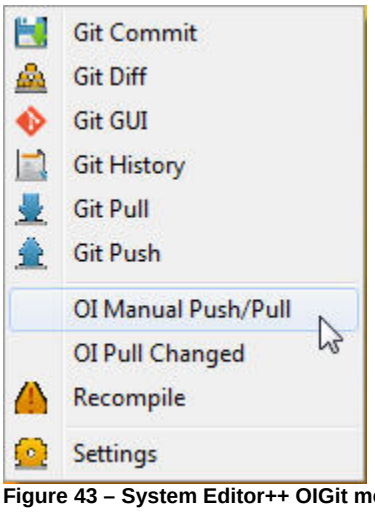

Figure 43 – System Editor++ OIGit menu – OI Manual/PushPull

This will open the OpenInsight Git window with the default “Push” operation and a selection type of “Current”

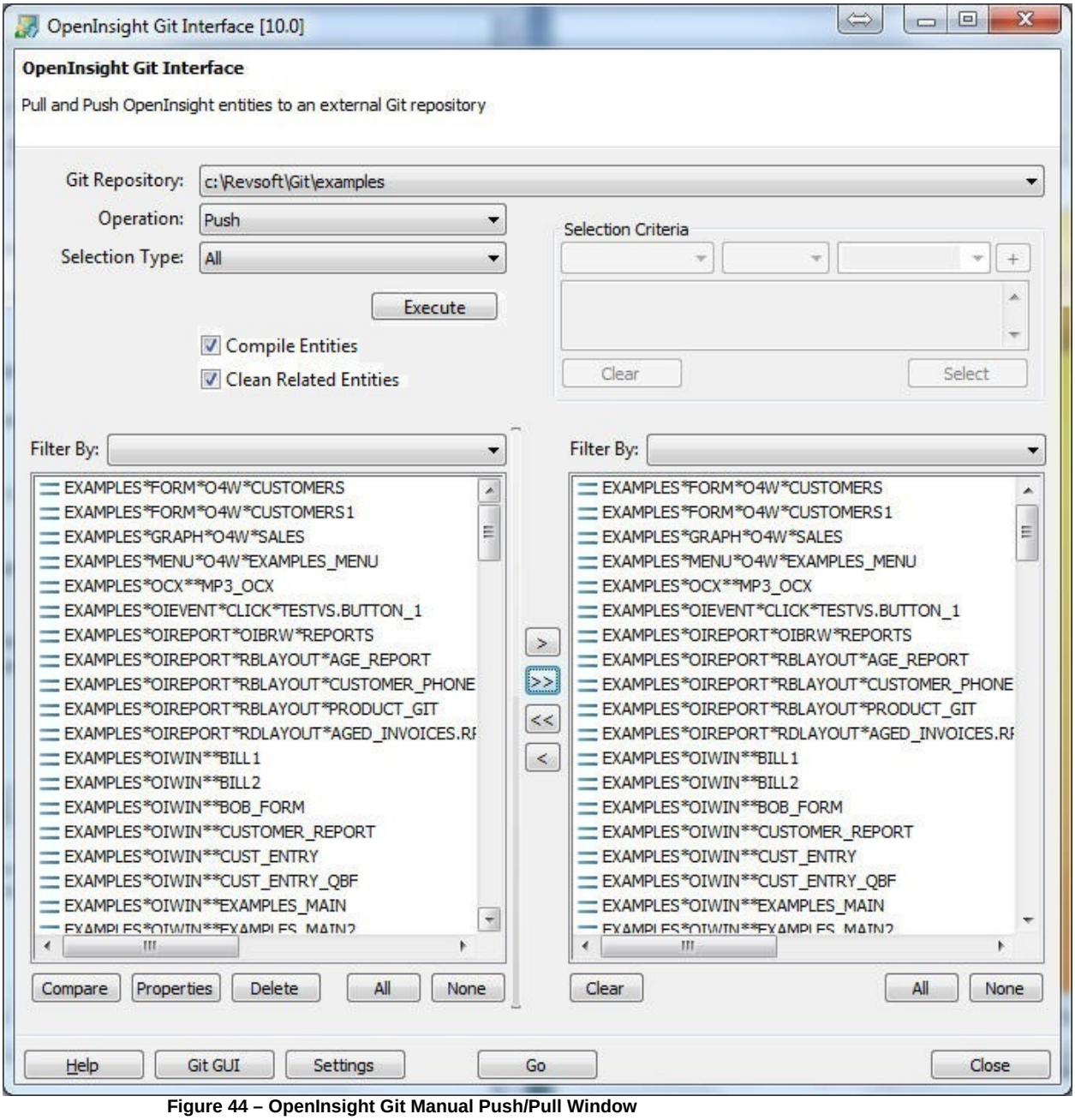

Figure 44 – OpenInsight Git Manual Push/Pull Window

The default Git repository for the current OpenInsight application will be automatically selected. See “Setting Up OpenInsight Git” for further details about maintaining the available Git repositories.

Push Selection type

There are 5 options for Push selection types:

a. All – Selects all publishable OpenInsight source entities.
b. Changed – Finds publishable entities in OpenInsight that are either different to the entity in the local Git repository or do not exist in the local Git repository.

c. Current – preselects the procedure or insert currently being edited, include any functions, subroutines or inserts that it may use.
d. Custom – Enables the “Selection Criteria” group box so you can select the types of entities you wish to push based on last update date and/ or last update user.
e. Since ddmmyyyy hh:mm – Changes since the last manual push date time. The selection is based on the last update date and time for an entity. The last manual push date and time are stored in the OpenInsight Git settings for the current application.

Entities that match the selection type will be listed in left list box. You can use the buttons between the list boxes to move entities in the left list box to and from the right list box. The entities listed in the right list box will be the ones “pushed” out to your local Git working directory. To push the selected entities click on the “Go” button. If you “push” an entity that is identical in OpenInsight and Git, Git will ignore the “push” for the entity.

If you wish, you can compare the selected entities in the left list box with the corresponding copy in your Git repository by clicking on the Compare button.

The icons next to each entity have the following meaning when performing a push operation.

	Does not currently exist in your Git repository, so it will be added
	You should never see this icon during a manual push process
	A version exists in Git. The push will update the Git version with the version in OpenInsight
	The file in Git is identical to the source in OpenInsight. These items are ignored during a manual push.

How to commit changes to your local Git repository

You can check if there are any changes that require committing by opening the OpenInsight for Git menu from the System Editor++ or opening your Git client software and clicking on the Commit button.

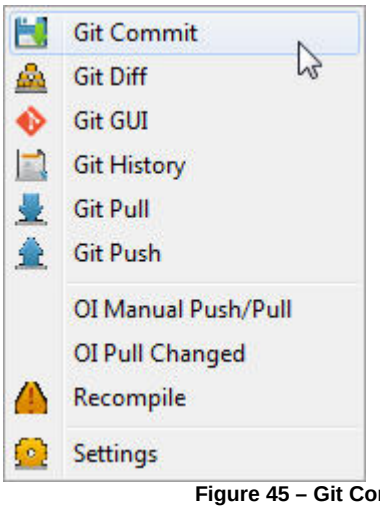

Figure 45 – Git Commit menu option

The Git Commit menu will only be enabled when there are uncommitted changes in your local Git working directory. The number of uncommitted changes will be displayed inside the square brackets.

Choose this menu option when you are ready to commit the changes to your local Git repository. This will open the Git Commit window.

The example window below is from the Git Extensions UI. Most Git UI tools will follow the same pattern.

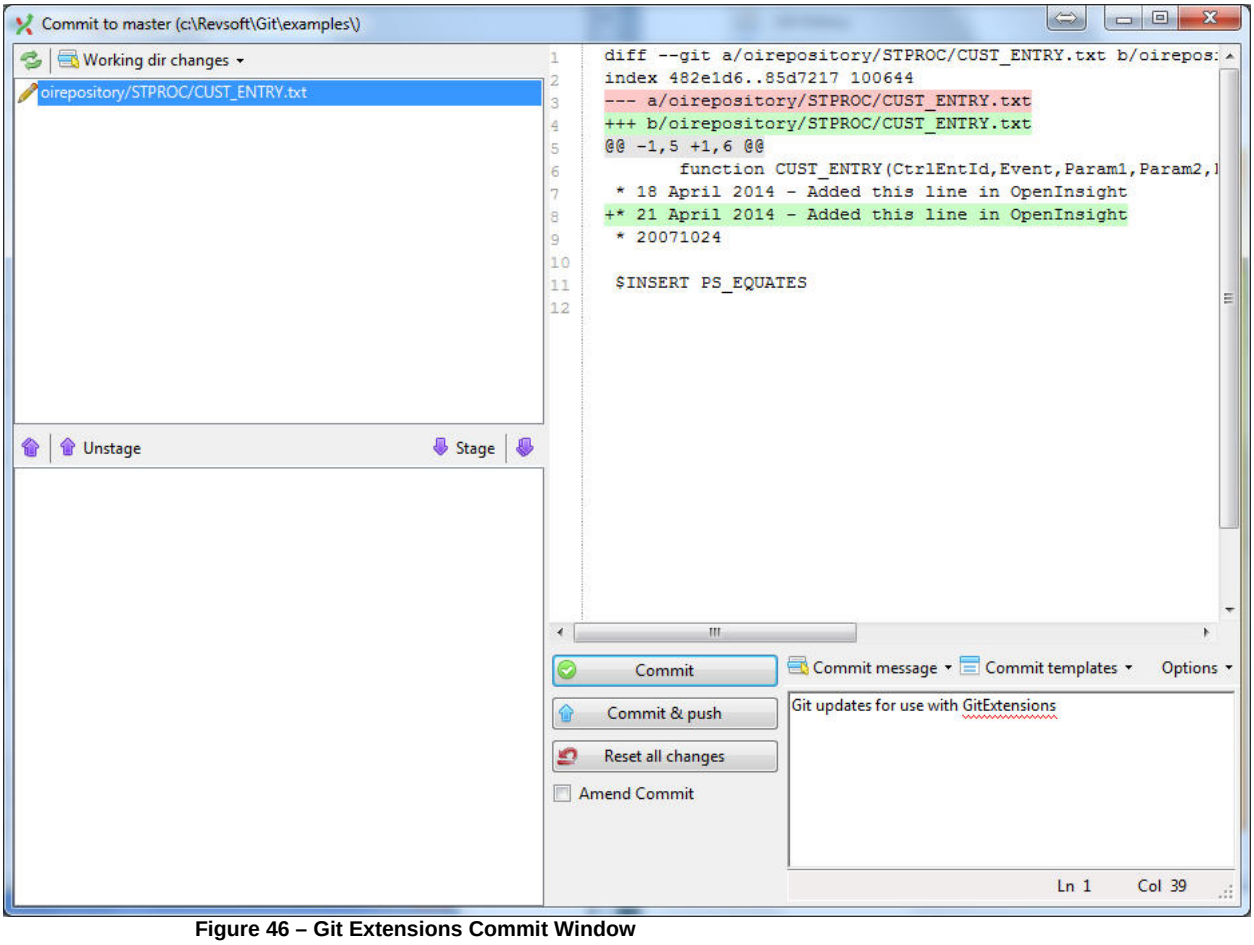

Figure 46 – Git Extensions Commit Window

1. The upper left box contains a list of unstaged changes to your local Git repository, most commonly referred to as your "working directory". You must stage the changes you wish to commit.
2. The lower left box contains a list of the changes you have staged and are ready to commit. Only the items in the staged box will be committed your main Git repository.
3. The upper right box displays the contents of the file currently selected in either the unstaged or staged boxes. When the item is in the staged box, this box contains the complete source code. When selected from the staged box, only the changes or deltas are displayed.
4. The lower right box is where you type a short description of the contents of this commit. e.g. updated Git to support Git Extensions client. The comment is optional but we strongly recommend that you write something short and meaningful. Other users will be able to read this comment, which will help them identify what changes occurred in the commit.
5. When you are happy with the selected changes and have typed in your comment, click on the Commit button. This will commit your changes to your local Git repository, commonly referred to as your "Master" branch.

   You can also click on the "Commit & Push button". This will commit your changes locally and then push them to your designed remote repository, commonly referred to as "origin". But, it is recommended that you always pull changes from the remote repository first. When you pull first, you resolve any conflicts locally, before you push your changes to the remote repository.

Pulling updates from a remote Git repository

It is recommended that you always push and commit your OpenInsight changes to your local Git repository before pulling changes from a remote Git repository.

If you are not using a remote Git repository, then you can skip this step.

To pull updates from a remote repository select the Git Pull menu option from the OpenInsight Git menu or open your Git Client software and select Pull.

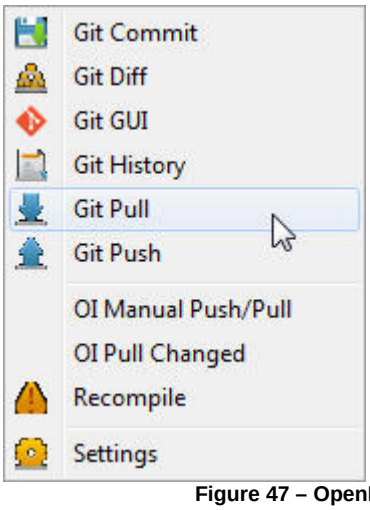

Figure 47 – OpenInsight for Git – Git Pull

The Git Pull window will appear. This example is from the Git Extensions UI.

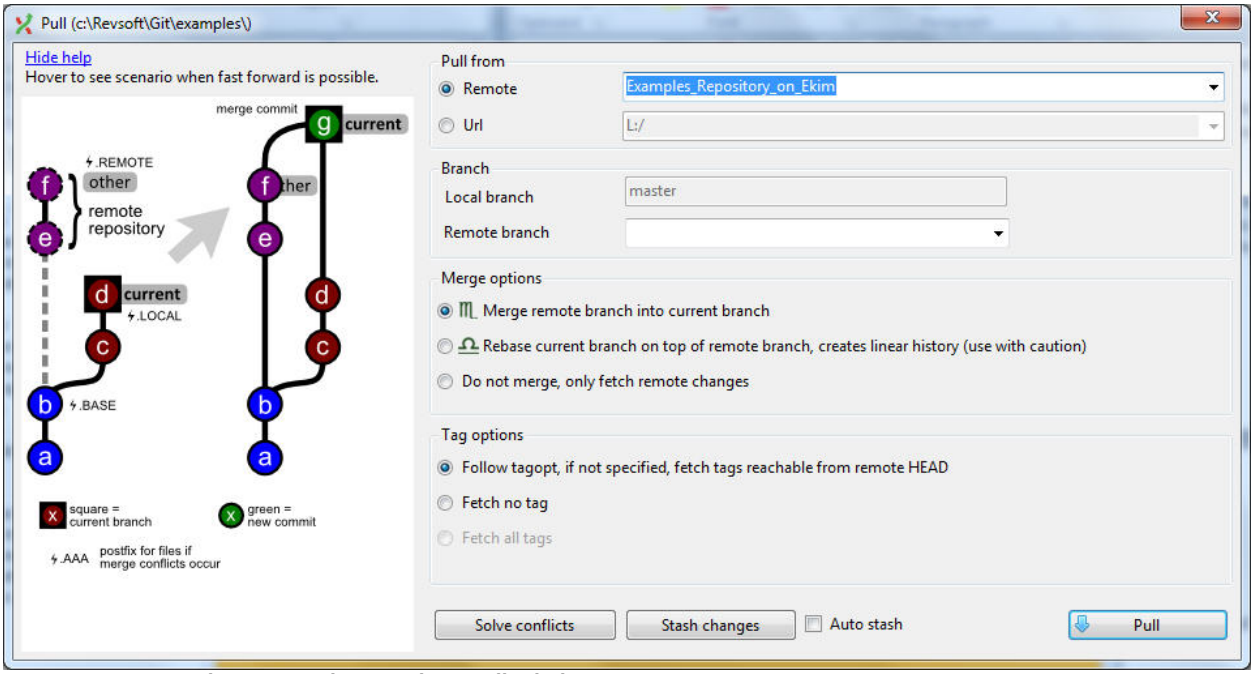

Figure 48 – Git Extensions Pull Window

(1) "Pull from" will automatically default to "origin", which is the remote repository you cloned from your local Git repository. You can change this to pull from any other remote repository using the Manage button.

(2) Select the “Merge remote branch into current branch” option, which is the default. This will pull updates from the remote Git repository and automatically merge them into your local repository. You will be notified if there are any conflicts during the merge process. You will need to resolve them before you can push/pull any changes from your local Git repository into OpenInsight or back out to any remove repository. See “Resolving conflicts” for more information.
(3) Click the Pull button to begin the Pull and Merge operation.

Once the Pull and Merge operation is complete and any conflicts are resolved, your local Git repository master branch will contain the latest version of the source code, including any changes pulled from remote repositories and any changes you made in your local copy of OpenInsight.

You are now ready to push your changes back to remote repositories.

Merge updates from remote repository with your local repository

When you pull updates from a remote repository Git does not automatically incorporate the updates into the master branch of your local repository. You must perform a “Merge” process to do this.

Some Git client tools, such as Git Extensions, can automatically merge the changes at the end of the “pull” process. This is a handy shortcut that saves you from performing a manual “merge” yourself.

The purpose of the merge process is to check for any conflicts between your local repository and the remote repository. Conflicts occur when the same line of code is changed in both your local repository and the remote repository copy of a file. If a conflict is found, none of the updates from the remote repository will be integrated with your local repository until the conflicts are resolved.

Below is an example of a conflict

Local Git copy	Remote Git Copy

`// loop to check a date`
`table.eof = false$`
`old.flag = false$`
`Loop`
`  readnext id else table.eof = true$`
`while id do`
`  old.flag = false$`
`  read record from table.handle,id then`
`  if record<17> < '12889' then`
`  old.flag = true$`
`  end`
`  end`
`repeat`
	`// loop to check a date`
	`table.eof = false$`
	`old.flag = false$`
	`Loop`
	`  readnext id else table.eof = true$`
	`while id do`
	`  old.flag = false$`
	`  read record from table.handle,id then`
	`  if record<17> < '16244' then`
	`  old.flag = true$`
	`  end`
	`  end`
	`repeat`

In the above code, the highlighted line has been changed in both the local copy and the remote copy of the file. During the merge process, this will be identified as a conflict. Any files containing conflicts will be listed in the merge conflict resolution window.

You must then resolve all conflicts using the Git client compare tool. Once all conflicts have been resolved you can then rerun the merge process.

Resolving conflicts

If any conflicts are found during the “merge” process, you will be required to manually resolve the conflicts before continuing.

Different Git clients offer different conflict resolution tools. Git Extensions offers a great tool called KDiff. KDiff presents both versions of the file side by side. You have the option of choosing either one as the master copy or you can merge parts of each file into a new file. See your Git client for further details on conflict resolution. You will not be able to perform any further commit, pull or push operations from Git until all conflicts are resolved.

Push updates to remote repositories

You should only push your changes to remote repositories once:

a) You have pushed and committed your changes from your OpenInsight system.
b) You have pulled and merged updates from the remote repositories and resolved any conflicts.

To push your changes to remote Git repositories select the Git Push option from the OpenInsight Git menu or open your Git client and select Push.

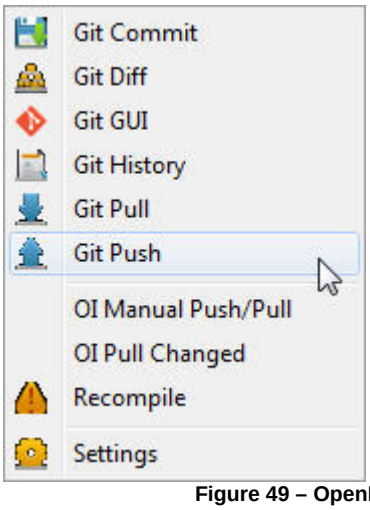

Figure 49 – OpenInsight Git menu – Git Push

This will open the Git Push client window. The example below is from the Git Extensions UI.

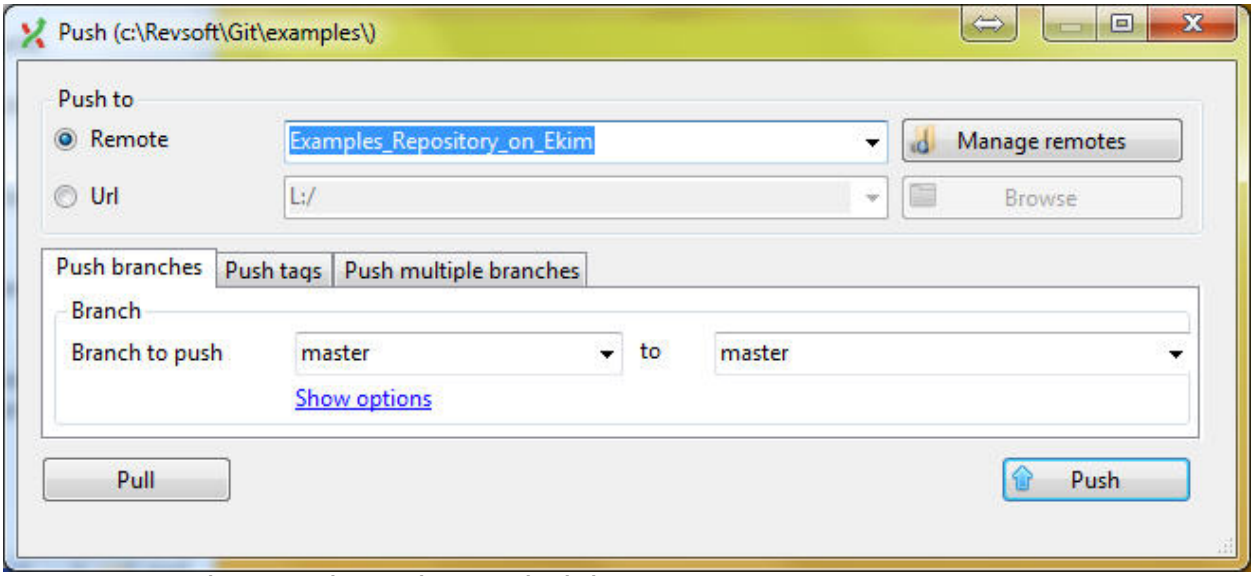

Figure 50 – Git Extensions – Push Window

(1) Select the remote Git repository you wish to push changes to. The default is "origin", this is the remote repository you cloned your local Git repository from. You can change this to push to any other remote repository using the Manage button.
(2) "Push Branches" should be set to master and master
(3) Click the Push button to push your changes to the remote Git repository. You will be notified if there any conflicts. Conflicts will need to be resolved before you make any further changes to your Git repository.

Once the Push operation is complete and any conflicts are resolved, the remote Git repository master branch will contain the latest version of the source code, including any changes you made in your local copy of OpenInsight and push and committed to your local Git repository.

Pulling updates from Git into OpenInsight

Pulling updates in from your local Git repository is the final step in the seven step process. The easiest way to do this is to select the OI Pull Updates menu option from the System Editor++ or open the OpenInsight Git menu from Application Manager, Application Tools and select the pull operation Pull Changed Selection Type.

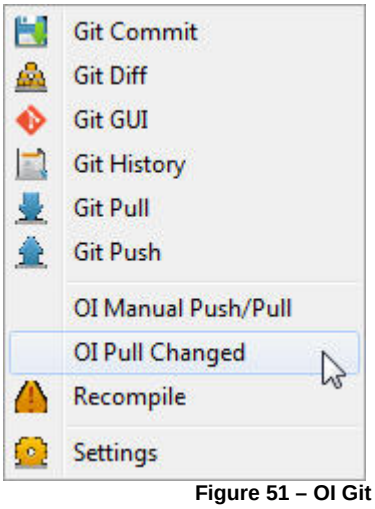

Figure 51 – OI Git menu – OI Pull Changed

This will open the standard OpenInsight Git window with the Pull operation and Changed Selection Types options preselected. It will also examine your local Git repository and identify and updates that need to be “pulled” into OpenInsight.

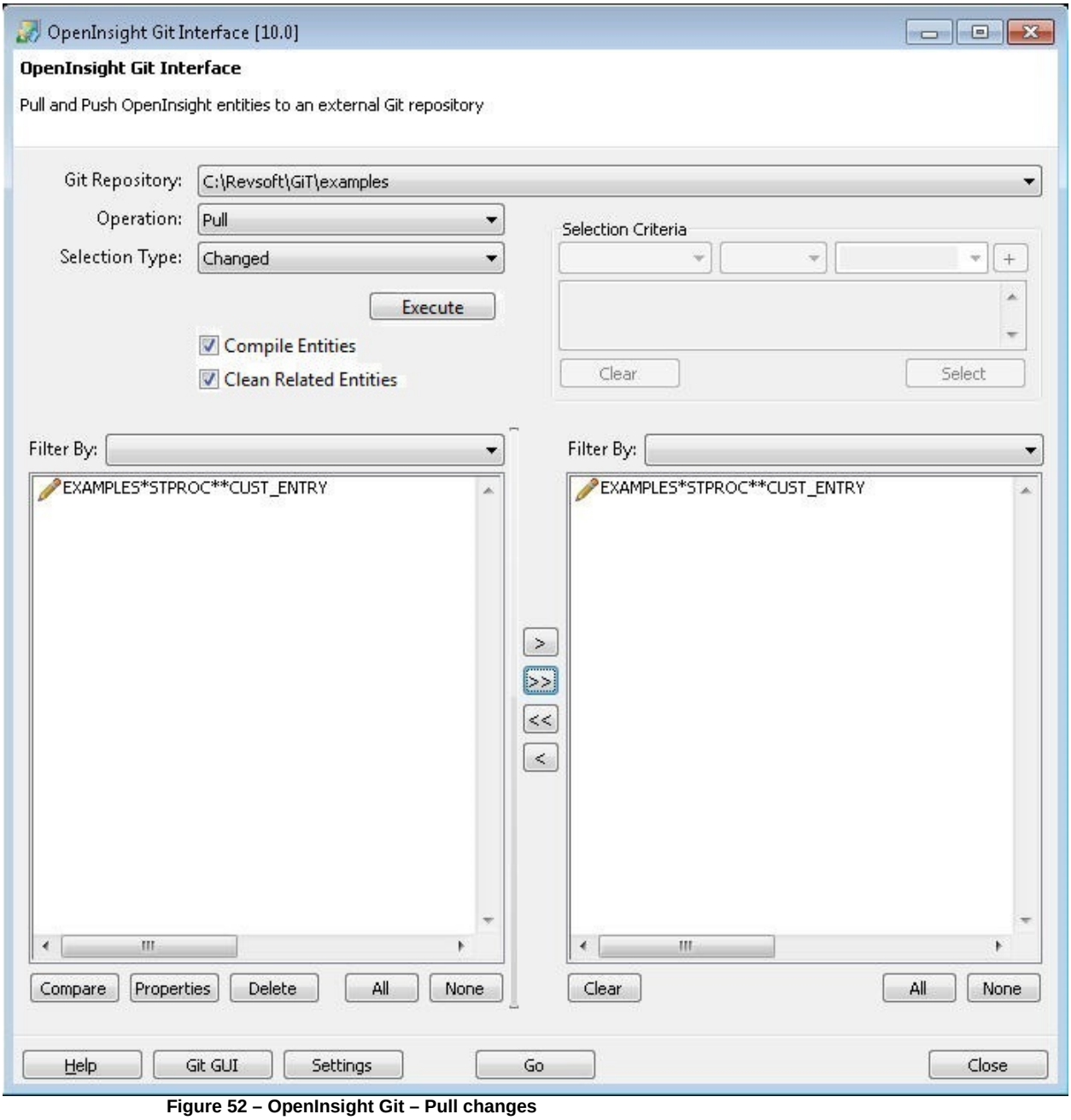

Figure 52 – OpenInsight Git – Pull changes

All Git entities that are different from their OpenInsight repository versions will be listed in the left listbox. Remember that this only means that they are different. In normal use, this usually means that the Git copy has been updated and you need to pull this update into OpenInsight.

However there may be times when your OpenInsight copy is more up to date than the Git copy, or you do not wish to pull in the copy from Git. In any case, you can select items in the left listbox and then click the Compare button to compare the OpenInsight version and the Git version.
Select the entities you wish to pull by moving them into the right listbox. Once you have selected all the entities you wish to pull, click the Go button. This will pull the selected entities into OpenInsight.

***Note:** Please ensure that you **do not have** any entities you wish to pull open in the editor or some other OpenInsight development tool. These entities will be locked and thus will not be available for update.*

The icons next to each entity in the list have the following means during a Pull operation

	Does not exist in your OpenInsight repository, so it will be added
	This OpenInsight entity has been deleted from the Git repository, so it will be removed from your OpenInsight repository (including any debug and executable versions)
	Exists in both Git and OpenInsight repositories, but the source in Git does not match the source in OpenInsight. You can use the compare button to compare these two versions.
	The file in Git is identical to the source in OpenInsight. These items are ignored during a manual pull.

Any OpenInsight windows, stored procedures and inserts pulled in will trigger a recompilation. In the case of inserts, all stored procedures in your OpenInsight repository that use the insert, regardless of where they have been updated or not, will be recompiled.

If a stored procedure in an inherited application uses the insert, it will be flagged as requiring recompilation. The next time you login to that application, you can select the Recompile menu option from the OI Git Menu.
If a recompilation fails, the item will be flagged as requiring recompilation. You can then perform a manual recompilation at a later time.

***Note:** Any OpenInsight procedures that are currently running (in the program stack) **cannot** be recompiled. Instead these will be marked as in need of recompilation.*

Any errors that occur during the pull process will be displayed at the end of the pull process. Typical errors include:

a) The entity is open in an OpenInsight development tool and thus is locked. Locked entities cannot be updated.
b) Recompilation failed – the compilation error code and description will be listed in the error log.

If a source entity fails to be pulled in successfully, check the error log for the error code and description. Once you resolve the error, you can try the Pull process again.

Recompiling procedures and windows

Usually when you pull in stored procedure, insert or window updates from your local Git repository, the necessary entities will be automatically recompiled. However, sometimes the recompilation may fail or it may not be possible to recompile at the time the updates are pulled in. For example:

a) The stored procedure is currently running in the program stack.
b) The stored procedure is in an inherited application.
c) The stored procedure failed recompilation for some reason.
d) The window failed to compile because the corresponding data tables were not attached or a dictionary field is missing.

When the automatic recompilation fails or is not possible, the entity is flagged as requiring recompilation. To check if you need to manually recompile entities, click on the OI Git menu and check the Recompile menu option.
If the Recompile menu option is enabled and has a number in the square brackets, this means that there are entities requiring recompilation. Select this menu option to display a list of entities to recompile.

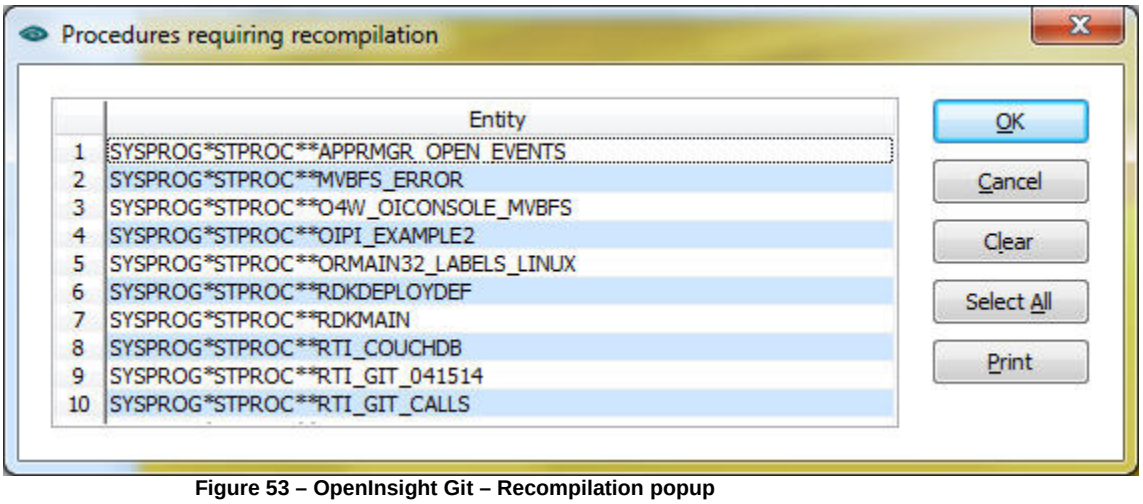

Figure 53 – OpenInsight Git – Recompilation popup

This lists all the entities in the current application that require a recompilation. Select the entities you wish to recompile and click the OK button.

An error log will be generated if any of the entities fail to recompile. The failed entities will remain flagged as requiring recompilation.

System Editor++ OpenInsight Git Menu

The OpenInsight Git Settings window controls which Git UI you wish to use.

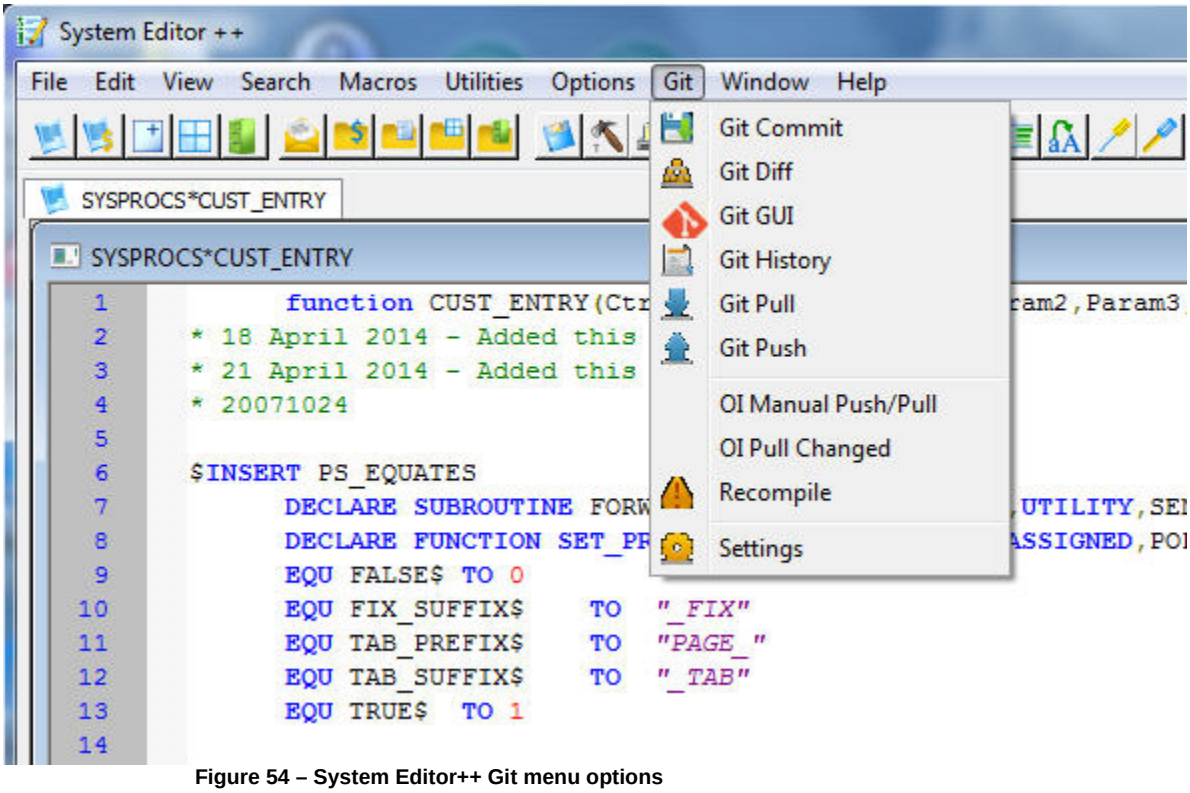

Figure 54 – System Editor++ Git menu options

Git Commit

This menu option displays the Git Commit window for your chosen Git UI. The menu option also displays the number of uncommitted updates in your working directory. If there are no uncommitted updates, this this menu option will be disabled. See “How to commit updates to your local Git repository” for more information.

Git Diff

The Diff menu option opens the Git UI Diff window. The Diff window lists each commit to your local Git repository including commits from other users that you pulled from the central Git repository. For each commit, Git Diff will list the entities that were updated in that commit. You can double click on an entity to view its complete history. See the “Using Git Diff Window” for more information

Git History

The Git History option opens the Git File History window for the current procedure or insert you are editing in the SRP Editor. The File History window displays the complete history of an entity, including all updates, and who changed them. See “Viewing Entity History” for more information.

Git Pull

The Git Pull option opens the Git Pull window. Use this window to pull updates from an external Git Repository into your local Git repository. See “Pulling updates from remote Git repository” for more information.

Git Push

The Git Push option opens the Git Push window. Use this window to push your committed updates to an external Git repository. See “Pushing updates to a remote Git repository” for more information.

OI Manual Push/Pull

This option opens the main OI Git window. From this window you can control what entities you wish to push or pull between OpenInsight and your local Git repository. See “Pushing updates from OpenInsight to Git“ for more information.

OI Pull Changed

This option opens the main OI Git window and preloads the window will a list of Git entities that are different to the entities in OpenInsight or do not currently exist in OpenInsight. This is the easiest way to pull updates from your local Git repository into OpenInsight. See “Pulling updates from Git to OpenInsight” for more information.

Recompile

The Recompile option displays a popup list of procedures or OI windows that require recompilation. The number of procedures or windows requiring recompilation will be displayed in square brackets. If no procedures or windows require recompilation, this menu option will be disabled. See “Recompiling Procedures and Windows” for more information.

Settings

Opens the Git Settings window where you can control different aspects of the OpenInsight for Git interface. See “Setting up OpenInsight Git” for more information.

REVELATION
SOFTWARE

Revelation Software, Inc
99 Kinderkamack Road Ste 109
Westwood, NJ 07675
U.S.A
Toll Free: 800-262-4747
Phone: 201-594-1422
Fax: 201-722-9815
www.revelation.com

Revelation Software Ltd.
Boundary House
Boston Road
London, W7 2QE
U.K.
Phone: +44 0 208 912 1000
Fax: +44 0 208 912 1001
info@revsoft.co.uk

BrightIdeas New Zealand
44 Cockle Bay Rd, Howick
Auckland, 2014
New Zealand
Phone: +64 9 534 9134
info@revelation.asia

Revelation Software is a division of Revelation Technologies, Inc.

Part No. 214-967